PROGRAMA DE DOCTORADO EN CIENCIA Y TECNOLOGÍA

TESIS DOCTORAL

ANÁLISIS DE PLANIFICABILIDAD Y OPTIMIZACIÓN DE SISTEMAS DISTRIBUIDOS DE TIEMPO REAL CON PARTICIONADO TEMPORAL

PhD THESIS

SCHEDULABILITY ANALYSIS AND OPTIMIZATION OF TIME-PARTITIONED DISTRIBUTED REAL-TIME SYSTEMS

AUTOR

ANDONI AMURRIO GONZÁLEZ

DIRECTORES

J. JAVIER GUTIÉRREZ GARCÍA

MARIO ALDEA RIVAS

UNIVERSIDAD DE CANTABRIA

Escuela de Doctorado de la Universidad de Cantabria

Santander 2022

This PhD. Thesis has been funded by the program
"Doctorados Industriales 2018"
and it has been done in collaboration between:

Ikerlan

Member of the Basque Research and Technology Alliance

&

University of Cantabria

Software Engineering and Real-time group

Author: Andoni Amurrio González

Ikerlan Supervisor: Ekain Azketa Ferreras

UC Supervisors: J. Javier Gutiérrez García
Mario Aldea Rivas

Abstract

Nowadays, most of the computing systems found in industry, such as autonomous driving platforms, smart grids or industrial control systems, are usually real-time cyber-physical systems that have a strong coupling between the software components, in charge of controlling the whole system by means of specific algorithms, and the distributed hardware equipment, such as sensing and actuating devices. In addition to their functional requirements, this kind of systems must meet complex non functional requirements related to execution predictability or fault tolerance, in order to comply with safety and application-specific standards.

In this context, designing real-time systems faces several challenges. Real-time systems are characterized by their deterministic response to external events, executing a sequence of activities with precedence relationships called end-to-end flows, which may be linear or multipath. Their correctness does not only depend on their results being computed correctly, but also on the time at which these results are provided, which is usually achieved by imposing deadlines that must be met even in the worst-case scenario. The implementation of sophisticated software architectures, such as time partitioning techniques that allow isolation among components, and the hierarchical combination of cyclic scheduling and fixed priority (FP) scheduling found for example in the avionics and space domains, requires the development of new response time analysis, scheduling and allocation techniques. All this has led to an increasing complexity in their design, forcing researchers from academia and industry to combine their efforts to bridge theoretical developments and practical use-cases. The work developed in this thesis targets a real railway signalling application, which is a paradigmatic example that brings together all the features just described.

With regard to the response time analysis for event-driven real-time systems, new approaches or techniques have been proposed in the literature based on models with different complexity, from single processor to multiprocessor or distributed systems, and dealing with linear end-to-end flows or other complex multipath logics. The vast majority of the underlying models for these techniques were based on the authors' conception about a hypothetical target system. However, current applications in industry show that state-of-the-art approaches need to be extended in order to calculate their worst-case response times accurately. In this thesis, a new technique

that captures the complexity of current applications, that is, time partitioning plus fixed priority scheduling and multipath flows, has been developed.

In addition, it is also essential to optimize the utilization of resources of the real-time application while guaranteeing that its deadlines are met. In the last three decades, a large number of proposals have been developed for the optimization of the scheduling and deployment of distributed real-time systems under different algorithmic approaches, which provide acceptable solutions for this problem categorized as NP-hard in the strong sense. A survey of these works has been reported in this thesis, and their solutions have been characterized.

Based on the proposed schedulability analysis technique, two crucial aspects of the scheduling optimization are addressed. On the one hand, a collection of eight algorithms for the priority assignment of multipath end-to-end flows has been proposed, which can be applied both to hierarchically-scheduled systems like the motivating industrial use-case, and also to general FP systems. Moreover, the scheduling optimization of time partitions is also addressed by proposing an algorithm to perform the assignment of partition windows for a fixed available utilization in each partition and, on top of this, another algorithm is proposed to optimize partition utilization.

Finally, a first step has been taken to solve the problem of allocating real-time applications to heterogeneous multiprocessor architectures. The allocation problem, which has been widely addressed in the literature, is far from being optimally solved due to the novel computing platforms that are being used nowadays in trendy domains such as autonomous driving or robotics.

All techniques proposed in this thesis have been implemented in several prototype tools, which has been used to process the experimental evaluation in the industrial use-case as well as in synthetic use-cases that represent a more general set of systems. These tools can be integrated within a Model-Driven Engineering (MDE) methodology for the development of safety critical real-time systems.

Resumen

Hoy en día, la mayoría de los sistemas de computación hallados en la industria, tales como la automoción, las redes eléctricas inteligentes o los sistemas de control industrial, son típicamente sistemas ciberfísicos de tiempo real, en los que hay una gran interacción entre los componentes software, encargados del control del sistema mediante algoritmos específicos, y el equipamiento hardware distribuido, como son los dispositivos de sensorización y actuación. Además de sus propios requisitos funcionales, estos sistemas deben cumplir complejos requisitos no funcionales relacionados con la predictibilidad de su ejecución o la tolerancia a fallos, para así cumplir con los estándares de seguridad genéricos y sectoriales.

En este contexto, el diseño de sistemas de tiempo real afronta varios desafíos. Los sistemas de tiempo real se caracterizan por su respuesta determinista a eventos externos, ejecutando una serie de actividades con relaciones de precedencia llamados flujos de extremo a extremo, en inglés *end-to-end (e2e) flows*, que pueden ser lineales o multitrayecto (en inglés *multipath*). Su rendimiento adecuado no sólo depende de que el resultado del cómputo sea correcto, sino también de que este resultado haya sido obtenido en un lapso de tiempo determinado, lo que generalmente se obtiene imponiendo unos plazos que deben verificarse incluso en el peor de los casos. La implementación de sofisticadas arquitecturas en el software, como por ejemplo el particionado temporal que permite el aislamiento temporal entre componentes, y la combinación jerárquica de los ejecutivos cíclicos y las prioridades fijas que se encuentran por ejemplo en la industria aeronáutica, requieren del desarrollo de nuevas técnicas de análisis, despliegue y planificación. Todo esto ha llevado a que su diseño sea cada vez más complejo, obligando a la comunidad investigadora, desde la academia a la industria, a unir sus esfuerzos para aplicar los avances teóricos en casos de uso prácticos. El trabajo desarrollado en esta tesis se centra en una aplicacion real del sector ferroviario, dado que se trata de un caso paradigmático que reúne todas las características expuestas.

En cuanto al análisis de los tiempos de respuesta de sistemas gobernados por eventos, nuevos enfoques y técnicas han sido propuestos en la literatura, basados en modelos de mayor o menor complejidad, desde sistemas con un solo procesador hasta sistemas distribuidos o multiprocesador, y y que tratan flujos de extremo a extremo (en inglés *end-to-end* o simplemente *e2e*) lineales como por otras arquitecturas multitrayecto

de mayor complejidad. La gran mayoría de los modelos y técnicas propuestos están basados en sistemas hipotéticos, fruto de la concepción de quien los propone. Sin embargo, las aplicaciones que se encuentran en la industria hoy en día evidencian que los métodos existentes han de ser extendidos para que se puedan calcular sus tiempos de respuesta de forma precisa. En esta tesis, se ha desarrollado una nueva técnica de análisis que captura la complejidad de las aplicaciones industriales, es decir, el uso de particionado temporal combinado con la planificación basada en prioridades fijas y flujos e2e multitrayecto.

Asimismo, también es esencial la optimización del uso de los recursos de una aplicación de tiempo real a la vez que se garantiza que sus plazos se cumplen. En las tres últimas decadas, se ha llevado a cabo una gran cantidad de propuestas para la optimización del despliegue y la planificación de los sistemas de tiempo real distribuídos, mediante distintos enfoques algorítmicos, que producen soluciones aceptables para este problema categorizado como NP-difícil. En esta tesis se presenta una revisión de estos trabajos y sus soluciones se han caracterizado.

Teniendo como eje la técnica de análisis de planificabilidad desarrollada, se abordan dos aspectos cruciales de la planificación de los sistemas de tiempo real. Por un lado, se propone una colección de ocho algoritmos para la asignación de prioridades para flujos e2e multitrayecto, que pueden ser aplicados tanto a sistemas con planificación jerárquica como el del caso de uso industrial que ha motivado este trabajo, como a sistemas distribuidos generales basados en prioridades fijas. Por otro lado, también se aborda la planificación de particiones temporales mediante un algoritmo de asignación de ventanas temporales para una utilización fija de cada partición, y sobre éste se propone otro algoritmo para optimizar la utilización de las particiones.

Finalmente, se dan los primeros pasos para resolver el problema del despliegue de aplicaciones de tiempo real en arquitecturas multiprocesador heterogéneas. El problema del despliegue, que ha sido ampliamente abordado en la literatura, está lejos de ser resuelto de manera óptima debido a la complejidad de las novedosas plataformas de cómputo que se emplean hoy en día en ámbitos como la conducción autónoma o la robótica.

Todas las técnicas propuestas en esta tesis han sido implementadas en herramientas prototipo y han sido empleadas para llevar a cabo la experimentación, tanto sobre el caso de uso industrial, como sobre experimentos sintéticos que representan casuísticas más generales. Estas herramientas pueden integrarse como parte de la metodología denominada ingeniería basada en modelos, en inglés *Model-Driven Engineering (MDE)*, para el desarrollo de sistemas críticos de tiempo real.

Acknowledgements

In these lines I would like to show my gratitude towards those that have supported me during this amazing journey.

One of the first days of work, I was told that the members of the ISTR group in the University of Cantabria were both lovely people and brilliant researchers. I can corroborate that at first hand, thank you so much for your kindness. I owe a huge debt of gratitude to my supervisors, Dr. J. Javier Gutiérrez and Dr. Mario Aldea, who have supported and guided me with wisdom, patience and perseverance.

To all members of the Dependable Embedded Systems group in Ikerlan, thank you very much for your guide and advice. I am deeply grateful to my supervisor Dr. Ekain Azketa, who is a reference for me and has worked really hard to carry out this thesis from the very first day.

I would also like to express my gratitude to Dr. Marko Bertogna, for accepting me at the University of Modena and Reggio Emilia. The incredible team at HiPeRT Lab made me feel at home, even during the hardest pandemic times, and I hope our paths will cross again. I must also thank Dr. Nacho Sañudo and Dr. Micaela Verucchi for their kind attention and advise during the stay, both in the office and beyond, and Carmelo Scribano for providing me with a safe and reliable means of transportation during my stay in Modena.

Euren maitasuna, beharbada merezi izan ez badut ere, egunero adierazi didatenak ekarri nahi ditut gogora, neurri handi batean lan hau haiena ere baita. Ehun mila esker familia eta kuadrilari, desberdintasunik balego bi hitz horien artean. Azkenik, bide luze honetan nire alboan uneoro sentitu zaituztedalako, aita, ama, Maddalen eta Ane, bihotz-bihotzez, eskerrik asko.

Nekez uzten du bere sorterria
sustraiak han dituenak.
Nekez uzten du bere lurra zuhaitzak
ez bada abaildu eta oholetan.

Contents

1 **Introduction** **1**
1.1 Context and background . . . 1
1.1.1 Real-time cyber-physical systems . . . 1
1.1.2 Real-time safety critical systems . . . 2
1.1.3 Model-driven engineering . . . 3
1.1.4 Response-time analysis . . . 4
1.2 Industrial use-case . . . 7
1.3 Objectives . . . 9
1.4 Organization . . . 10

2 **Scheduling and optimization in distributed real-time systems: a literature review** **13**
2.1 Introduction and methodology . . . 13
2.2 Genetic Algorithm (GA) . . . 15
2.3 Tabu Search (TS) . . . 18
2.4 Simulated Annealing (SA) . . . 19
2.5 Mathematical Programming . . . 21
2.6 Branch and Bound (BB) . . . 25
2.7 Heuristics (HEU) . . . 26
2.8 Classification of works and conclusions . . . 31

3 **Real-time system model** **35**
3.1 Logical architecture . . . 35
3.2 Physical architecture . . . 37
3.3 Hierarchical scheduling . . . 38
3.4 Sensitivity analysis . . . 39
3.5 Modeling the industrial use-case . . . 42

4 **Response-time analysis** **45**
4.1 Response-time analysis of linear e2e flows . . . 45
4.2 Response-time analysis of multipath e2e flows . . . 46
4.2.1 Simple example . . . 47

4.2.2 Implementation and tools 48
4.3 Industrial use-case evaluation 49
4.4 Response-time analysis performance 51
4.5 Conclusions 54

5 Priority assignment 57
5.1 Scheduling-parameter assignment overview 57
5.2 Priority assignment in multipath e2e flows within time partitions . . 60
5.2.1 Virtual Deadline assignment 60
5.2.2 Virtual Deadline transformation into priorities 68
5.3 Evaluation of the priority assignment algorithms 70
5.3.1 Industrial use-case 70
5.3.2 Performance evaluation 73
5.4 Conclusions 80

6 Partition window assignment 81
6.1 Study of the influence of partition windows on schedulability 81
6.1.1 Available Utilization 82
6.1.2 Number of windows 82
6.1.3 Context switch overheads 83
6.1.4 Conclusions of the study 85
6.2 Heuristic partition window assignment 86
6.2.1 Window Assignment (WinAs) Algorithm 86
6.2.2 Heuristic Optimized Partition Window Assignment (HOPWA) 89
6.3 Performance evaluation 91
6.3.1 Design of the synthetic experiments 92
6.3.2 WinAs algorithm characterization 95
6.3.3 Evaluating HOPWA algorithm 99
6.3.4 Scheduling evaluation of the industrial use-case 106
6.4 Conclusion 107

7 Step-to-processor allocation 109
7.1 Background 109
7.1.1 Multicore achitectures 110
7.1.2 Allocating real-time applications in heterogeneous systems . . 111
7.2 Slack-Based Allocation (SBA) algorithm 112
7.3 Preliminary evaluation 114
7.4 Conclusions 117

8 Conclusions 121

8.1 Thesis contributions 121
8.2 Future Work 123
8.3 Publications 124

Bibliography **127**

1 Introduction

In this chapter the context of this thesis is described, explaining the most relevant properties of real-time cyber-physical systems, real-time safety-critical systems, response time analysis and model-driven engineering, which are fundamental concepts in this work. Then, the motivational railway signalling application is presented as a paradigmatic industrial use-case of the schedulability analysis and optimization problems addressed here. Finally, the objectives of this thesis are listed, and the organization of this document is described.

1.1 Context and background

1.1.1 Real-time cyber-physical systems

In real-time systems, correct function depends not only on results being right, but also on them being produced in time, meeting the timing requirements imposed on the software, usually deadlines. Nowadays, many of these systems correspond to cyber-physical systems, which are information technology systems integrating computation, storage and communication capabilities along with sensorization and/or control of elements or devices in the physical world. This type of systems, as well as the technology developed around them, is particularly relevant in the conceptual framework of Industry 4.0 (or the fourth industrial revolution) on which both research entities and businesses are currently focusing their attention.

From the viewpoint of their physical architecture, these cyber-physical systems are in reality distributed real-time systems, with several processors that may be homogeneous or heterogeneous in terms of their core architecture and computation speed, connected by one or more communication networks. Such networks may be based on different standards, such as the IEEE 802.1 working group's Time Sensitive Networks (TSN) [TSN] or partitioned such as ARINC-664 [Aer09] from the avionics domain. Their logical architecture is composed of tasks and messages. Both are characterized by deadlines, which are timing requirements that must be met even in the worst scenario, and by worst-case execution/transmission times. Their activation can be periodic or not. The task constitutes the minimal schedulable unit in a computer, which can only be executed if the required hardware and software resources are available. Their activation can undergo some variability, which is known as jitter. The messages, whose minimal schedulable units are the packets, enable the communication among tasks with precedence relationships, making up the so-called end-to-end flows (henceforth e2e flows). All tasks and messages are activated when certain stimuli are produced. Depending on the nature of the stimuli, two criteria for classification of the real-time systems are recognized in [Kop11]: those activated by time and called Time-Triggered (TT), and those activated by events and called Event-Triggered (ET).

Scheduling the execution order of concurrent tasks is determined by an algorithm implemented in a component called scheduler, which is in charge of determining when tasks are executed in a processor (and also when messages are sent through a network). Determining in which computers the tasks are executed and through which networks the messages are sent, often known as allocation, increases the problem's complexity, thus making it an NP-hard problem [TBW92]. Algorithms providing optimal solutions in polynomial time for this type of problems are not known yet, so typically generic search and optimization algorithms are used.

1.1.2 Real-time safety critical systems

A critical system is one that has functions whose failure can bring about severe consequences for humans, materials and/or the environment. One of the main characteristics that a critical system must provide is dependability, bringing together the concepts of availability, reliability, integrity, maintainability and safety. Safety is quantified through the "Safety Integrity Level" (henceforth SIL), which is the relative level of risk reduction provided by a safety function, or an objective level for risk reduction [SS04]. The safety levels range from SIL1, the lowest, up to SIL4, the highest.

Complex systems such as automobiles, trains and airplanes combine non-safety functions with other different-level safety ones. In the past, these functions were deployed in independent physical systems to avoid interference of the non-safety ones with the safety ones. This paradigm usually requires a large amount of heterogeneous equipment, leading to increased costs of installation, start-up and maintenance.

With the aim of reducing costs and using resources efficiently, cyber-physical systems have evolved in recent years by incorporating multi-processor architectures and also virtualization techniques, thus enabling the execution of distinct functionalities of the system in spatially and temporally protected environments, called partitions, on a single hardware platform. Each partition can have distinct non-functional requirements to guarantee response times, safety, confidentiality, etc., which make up the so-called mixed-criticality systems [Ves07]. In [BD17], a detailed review of the work done in the last decade on this type of systems is carried out, ranging from the most theoretical scheduling or design aspects, to the basic implementation mechanisms. In mixed-criticallity systems it is necessary to have total independence among partitions with the aim that the processes of specification, design, implementation, certification (in those systems that require it) and execution are totally independent throughout the distinct system components [SO12] [BLS10] [Goo+13] [Cre+14]. In the European project MultiPARTES [Tru+14] for example, a set of tools was proposed for the development of mixed criticality systems based on partitioning, from hardware and software architectures to partition management tools.

In safety-critical systems, real-time operating systems are typically used to guarantee the deterministic execution of tasks, based on a specific scheduling policy or a combination of them, including cyclic executives, fixed priorities or deadline-based priorities [Liu00]. Cyclic executives were the most widely used until the advent of priority-based scheduling, which has had an enormous repercussion in most designs [PEP04a], and which is present in operating systems that follow the POSIX [IEE03] or AUTOSAR [AUT03] standards, in communications networks such as CAN [Bos91] and in programming languages such as the Ada standard [ISO12]. Nowadays, partitioned systems enable the combination of characteristics of cyclic executives and fixed priorities in a hierarchical scheduling, with the partitions as the basic scheduling level and the use of priorities within each partition. This is the case of VxWorks [Win16] used in avionics, or Integrity [Gre] used in the railway domain, which are SIL3 certified.

1.1.3 Model-driven engineering

The methodology consisting of creating abstractions, i.e. models, that allow developers to capture key information which is essential to the system-development process is called Model-Driven Engineering (MDE) [Sch06]. The abstraction of a system into a model allows developers to focus on relevant information independently from the platform where it is implemented. There are several aspects to address during the development of cyber-physical real-time systems, such as timing features, safety and security issues, logical and physical architectural designs, etc. All the information contained in those aspects, represented by different models capturing different views of the same system, is formalized and then it can be processed through a series of model transformations, which greatly facilitates development.

The Object Management Group (OMG) defined several concepts for modeling real-time systems, gathered in the MARTE [Obj11] standard. In this context, the University of Cantabria developed the MAST model [Gon+01] to capture the time-related features of real-time systems, modeling the tasks and messages that compose the e2e flows in order to analyze them via response-time analysis techniques. Its first version, which allows both single-processor and distributed systems to be modeled, is used in all the analysis and optimization tools within the MAST tool suite. Its second version, MAST 2 [Har+13], changed a few element names in order to align them with MARTE's UML profile [Obj11] for real-time embedded systems, and also includes hierarchically scheduled systems, support for network switches and other novel scheduling approaches. MAST's implementation is open source under GPL license and it has been successfully integrated in other model-based development tools, such as TEMPO [Hen+15], as part of the MDE methodology.

1.1.4 Response-time analysis

Guaranteeing the correct timing behavior of a system, even after building it, cannot be done generally by testing, as it is not possible to assure that the worst-case scenario has been considered by means of this methodology. Response-time analysis techniques are used to calculate the worst and best-case response times of tasks in a real-time system. This subsection contains a brief introduction to the most relevant approaches that have provided the basis for the new technique developed in this thesis.

Response time analysis for distributed real-time systems was first addressed in [TC94], where the authors propose the so called *Holistic* analysis for fixed priority systems. In this approach, each processor is analyzed independently, and it is assumed that all tasks are activated at the same time, thus not considering precedence relations that characterize e2e flows. That activation time is called the critical instant and leads to the busy period, which is the time when the processor is executing tasks with higher or equal priority to the task under analysis. This technique, rather than being exact, which would be intractable for complex systems, obtains upper bounds of worst-case response times. In that work, deadlines could be greater than the activation rates, and the activation *Jitter* that workload events may suffer was incorporated into the worst-case analysis.

With the aim of reducing the pessimism of Holistic analysis, offset-based analysis techniques were proposed [Tin94]. Offsets express the minimum instant at which a task is activated, which leads to a less pessimistic worst-case response time calculation. Later, this technique was extended to distributed systems by [PG98]. In [PG99] an improvement of the offset-based technique was introduced, which takes into account the precedence relationships of tasks to provide a tighter estimation of the response times. Another improvement of the offset-based technique was proposed in [MN08], which provides better results than [PG98] but it does not always improve the results of [PG99].

In [Pal+16] the authors propose an offset-based analysis technique for distributed real-time systems based on time-partitioning, based on the compositional approach of [Riv+11]. In order to analyze time-partitioned systems, each partition is analyzed independently. The rest of the partitions are modeled as a high priority e2e flow, called *Unavailability flow*, which is integrated in the offset-based analysis. This unavailability flow is inspired by the concept of *availability function* introduced in [AP04] to model the partition windows during which a given task may be scheduled; the authors used the inverse of this function for the analysis.

Modern real-time applications usually exhibit complex activation patterns, where a task may trigger more than one task, i.e. a fork pattern, and similarly, a task may be activated after the execution of one or more tasks is completed, i.e. join/merge patterns. This model is commonly known as a multipath model. In addition to the aforementioned MAST model, based on the MARTE standard´s e2e flows, the Directed Acyclic Graph (DAG hereafter) model is a system model that can also describe multipath architectures, which is widely used in the real-time community. It was first introduced by [LA10] to support the analysis of real-time tasks, and in [LA11] it was extended to distributed real-time systems.

The schedulability analysis of multipath e2e flows was addressed in [GPH00], where the authors described the procedure to calculate the worst-case response times of tasks within multipath e2e flows. This technique is based on the holistic approach, which is known to be pessimistic. In [HE05] a compositional approach is proposed called Symta/S, where systems are modeled as networks of resources and workloads as tasks with precedence constraints. However, only fork patterns, and not join ones, are considered in this approach, which they name tree-shape dependecy model. In [Fon+16] a schedulability analysis method for DAG tasks is proposed, although it is not applicable in the context of this thesis due to some limitations that will be discussed later.

To this day, there is no response-time analysis technique that includes the two fundamental features addressed in this thesis: time-partitioning and a multipath model.

There are other approaches that have addressed response time analysis considering similar system models. For instance, in [KHB16] a timing analysis approach for cyclic real-time stream processing applications is presented. The targeted architectures are multiprocessor systems. Another interesting contribution on the schedulability analysis of time-partitioned systems was proposed in [Mar+12], where a response time analysis method is proposed for mixed-criticallity applications. This technique is not applicable to this work since it does not allow tasks of the same e2e flow to be located in different partitions, which is a common feature in partitioned systems that make use of an Input/Output (I/O) partition to handle communications with other nodes of a distributed architecture.

A special mention should be made of Assertion Based Verification techniques, which include formal verification methods where designs are verified against certain assertions. In [Anw+20] a unified framework for executing static and dynamic verification [Anw+19] for embedded system design is presented. One common static verification method is Timed Automata Models [AD94] which enable the determination of whether or not timing requirements are met. A popular tool that is used in this context is UPPAAL [Hes+08], which can be used to assess whether a system will meet its timing constraints. A recent research work [HZZ20] shows how ARINC-compliant time-partitioned schedulers can be modeled following this method. However, the response time analysis techniques explored in this thesis have several advantages that may be important in some classes of systems. On the one hand, the result of the analysis, i.e., worst-case response times, are very intuitive numbers for engineers who want to assess how far or close the response times are to the deadlines. On the other hand, response time analysis can be used easily to analyze complex systems, which may have rather large timed automata models. Besides, for complex models response time analysis is considerably faster than applying model checking techniques. Readers are encouraged to read [Per+09], where the authors perform a deep study on how different abstractions affect the performance analysis of real-time systems, concluding that there is no abstraction that always outperforms the others.

These reasons motivate the extension of response time analysis to be applied in hierarchical time-partitioned systems to multipath flows.

1.2 Industrial use-case

The use-case addressed in this thesis challenges the state-of-the-art schedulability analysis and scheduling optimization techniques developed so far, applied to safety-critical distributed real-time systems. It is based on a railway signalling application, part of the European Rail Traffic Management System (ERTMS) [ERT06]. ERTMS is a standard resulting from an important European industrial project which aims to create a common system for traffic signalling and management in railways. This standard has two main components: GSM-R (Global System for Mobile Communications - Railway), which is in charge of wireless communications, and ETCS (European Train Control System), which performs signalling and supervision duties for traffic management. On-board ETCS is the distributed and safety-critical equipment within the vehicles that performs computation tasks. It is connected to other on-board subsystems, such as switches, sensors or other processing units performing secondary functionalities, by point-to-point connections through interfaces specified in the standard.

Trains receive driving indications and restrictions from the railway infrastructure, such as balises and other interfaces, for instance radio connections with centralized control centers. The main duty of the on-board ETCS subsystem is to provide drivers with all the information needed for safe driving, as well as supervising that the train is travelling respecting the received instructions at all times.

To do so, different safety functionalities must be carried out within a bounded time. In this work, the signalling application performs three functionalities: (1) Applying the Emergency-Brake (EB functionality), (2) Radio Block Center communication session establishment (RBC-CS functionality) and (3) Parameter visualization in the Driver-Machine Interface (PV-DMI functionality). Each of these functionalities, as well as related software and hardware, need to be certified for a certain integrity level. Safety standards force all safety functionalities to be SIL4, while other non-critical functionalities might be lower, even though they share the execution platform with high-criticality ones. On the other hand, execution platform suppliers only guarantee SIL 2 hardware equipment, so another mechanism must be implemented in order to obtain SIL4 functionalities, even when software has been certified for SIL4. This mechanism is, in this case, a Dual Modular Redundancy (DMR) architecture, where results are voted for following a 1oo2 scheme [IEC10]. Event synchronization therefore becomes a major concern, since higher integrity levels can be reached with replicated software architecture rather than using a single instance of the execution platform.

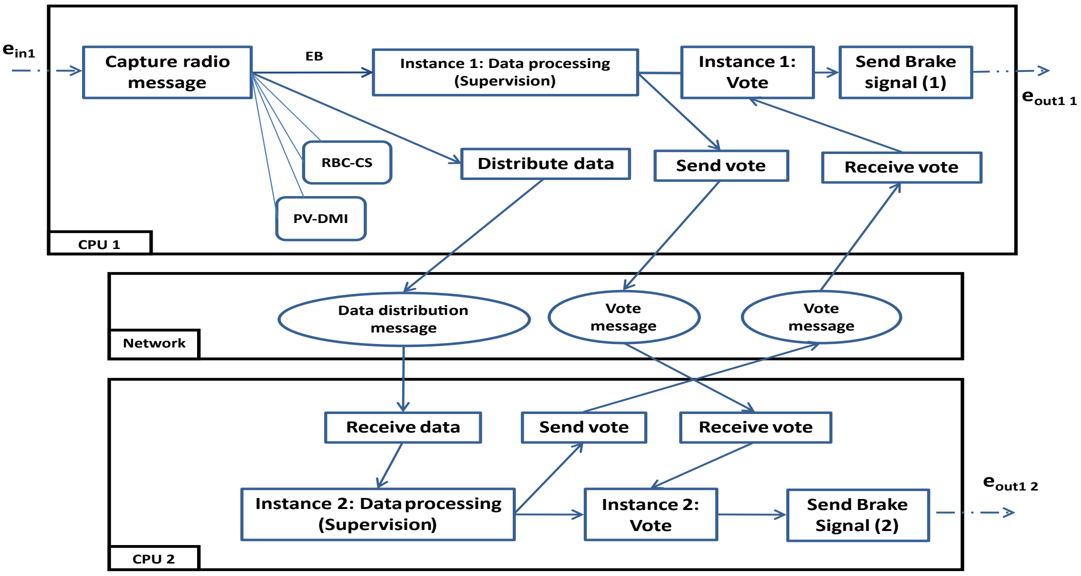


Fig. 1.1: Architecture of EB functionality

Figure 1.1 describes in detail the multipath architecture of the EB functionality for a SIL4 implementation. The other two functionalities (RBC-CS and PV-DMI) follow the same architecture. A workload event, which is triggered when the train runs through a balise or receives a radio message, activates a sequence of activities that are briefly explained as follows: (1) the message coming from any interface, considering that its integrity level is guaranteed by previous processing, is captured, (2) supervision is then performed at the first instance in CPU-1 by processing the captured data, and (3) concurrently, data is distributed to the second instance executing in CPU-2 for redundant processing, (4) results obtained from both supervision functions are interchanged, so that they are independently voted for each processor, and (5) the voting results are sent twice to the external subsystem in charge of commanding the brakes. Deadlines are imposed on the brake activation events ($e_{out1\ 1}$ and $e_{out1\ 2}$) referred to the signal reception (e_{in1}). Missing these timing requirements would lead to a system failure and a train crash might happen.

Train manufacturers have already implemented such an application. However, its implementation is based on a simple cyclic executive where all functions composing functionalities are called periodically, even if they do not have useful work to do. This results in inefficient resource usage which has become a major concern for train companies: 100% of processing capacity is dedicated to the execution of the application. The current implementation does not support the addition of extra functionalities easily or the possibility of integration it with other applications while guaranteeing the system's integrity. Worst-case response times are estimated now by testing the sequential application code to ensure that deadlines are met, and integrating this application with others would require more sophisticated response time analysis techniques than just testing. For these reasons, manufacturers seek a complete system re-design, making use of modern techniques such as those used in avionics or the automotion industry mentioned in the previous section. Timing behavior prediction of critical parts is essential. However, and to the best of our knowledge, there is no response-time analysis technique that could be directly applied to the redundant application based on partitioning addressed in the use-case, in order to check whether timing constraints are met. Therefore, in parallel to the system re-design, a new schedulability analysis technique will be developed, as well as an implementation in an analysis tool that will enable the validation of timing constraints of applications following the new architecture. Then, new optimization algorithms based on this approach will be proposed, including priority assignment, partition window assignment and allocation strategies.

1.3 Objectives

The general purpose of this thesis is, based on the requirements that have been identified in the aforementioned use-case, to investigate optimized techniques for the schedulability analysis and optimization of partition-based distributed real-time systems with multipath flows. It is common that these systems are found in safety critical domains, such as automotive or avionics, where cyber-physical systems must adhere to strict safety requirements imposed by certification authorities. That is why the developed techniques, devoted to guaranteeing that timing requirements are met, are intended to be applicable to other systems with analogous requirements.

To do so, the following particular objectives will be addressed:

- Formalisation of the system model. It should be a realistic system model capable of capturing all the time-related features that describe the timing behavior of real-world cyber-physical systems used nowadays.
- Development of a schedulability analysis technique for partition-based distributed real-time systems with multipath flows, which enables the determination of whether the imposed deadlines are met under specific scheduling schemes. This technique will be implemented complying the system model previously mentioned, which will improve the precision of existing approaches.
- Development of algorithms that carry out the synthesis of the targeted systems, based on the proposed response-time analysis:

    - Priority assignment algorithms: Based on the state-of-the-art algorithms, several non-iterative priority assignment algorithms will be proposed and adapted to the system model addressed in this work.

    - Partition window assignment algorithm: An algorithm that produces a partition window assignment for each partition will be developed.

    - As a secondary objective, an allocation algorithm will be developed in order to address the allocation problem in heterogeneous architectures. Providing a complete solution to the industrial use-case is the priority of this thesis, where the allocation problem is not fully related to it.

- Validation of the proposed solutions. The aforementioned railway signalling application will be targeted, and general and representative systems will also be evaluated, in order to characterize the behavior of the proposed algorithms.

As a result of this thesis, it is expected that a methodology for developing distributed and partition-based real-time systems will be formalized. This methodology will include modeling, analysis, deployment and scheduling phases, and it will be composed of several tools that will be used in real-time safety-related industrial applications.

1.4 Organization

This thesis is organized as follows. After this Introduction, where the most relevant concepts of this work are presented, an extensive literature review regarding the optimization of distributed real-time systems is detailed in Chapter 2.

Chapter 3 presents the system model that has been selected to describe the time-related features of the applications addressed in this thesis.

In Chapter 4, the development of a new response time analysis technique is presented, together with its application and evaluation on the target industrial use-case and also to general distributed real-time systems scheduled by FP.

Chapter 5 describes the priority assignment algorithms proposed in this thesis, which can be applied to the railway use-case and also to general FP distributed systems. Some conclusions referred to their applicability are drawn in this chapter.

Chapter 6 contains the partition window assignment algorithm, which is used to derive the primary scheduler of the hierarchical schedulers addressed in this work.

In Chapter 7, the allocation problem is introduced and first steps are given, providing some solutions and opening the path towards more sophisticated ones for this open problem.

Finally, the most relevant conclusions of this thesis are drawn in Chapter 8, reviewing the objectives set at the beginning of the work. In addition, future research challenges to be addressed from now on are presented.

2 Scheduling and optimization in distributed real-time systems: a literature review

This chapter compiles the work carried out on the task allocation and scheduling of distributed real-time systems. The works are classified according to their algorithmic approach and a brief outline of each one is provided.

2.1 Introduction and methodology

The comprehensive literature review carried out in this chapter contributes to two main aspects. On the one hand, to complement the work [BD17] in which a detailed review was carried out of mixed criticality systems focused on the temporal analysis, both of single-processor and multiprocessor systems, and in which different system models and real industrial applications are also described. This work complements that review in relation to the optimization of the allocation and scheduling of distributed real-time systems, aspects that were dealt with in less depth and that are of great interest in the development of current systems. On the other hand, it also considers other works on allocation and scheduling optimization, which, although not focused on mixed-criticality systems, do address distributed real-time architectures that are of interest in the design of current industrial applications.

Thus, this chapter compiles works considering this problem, since it was approached for the first time in [TBW92] up to the first half of 2021, mostly ordered chronologically and classified according to the algorithmic focus employed. All the works are included in a table showing their most remarkable aspects, with the aim of providing a clear summary of the state-of-the-art and facilitating consultation for future work on the optimization of the allocation and scheduling of distributed real-time systems.

As mentioned in the previous chapter, due to the NP-hard nature of the problem of allocating and scheduling of distributed real-time systems, optimization algorithms are used. These provide valid solutions that may be optimal or sub-optimal. This chapter focuses on works that use different algorithmic approaches, which will be briefly described:

- Genetic algorithms (GAs)[Hol75] are generic probabilistic metaheuristics that form part of the so-called evolutionary algorithms. These algorithms imitate biological mechanisms (adaptation to the environment, mutation, crossover, inheritance ...) that guide the evolution process in species, and which are used to look for solutions to diverse problems in wide search spaces.

- Tabu Search (TS) [Glo86] is a general-purpose metaheuristic procedure for search and optimization, which forms part of the local search techniques. Its main characteristic is the possibility to avoid sub-optimal local solutions within the solutions space of a specific problem.

- Simulated Annealing (SA) is a generic probabilistic metaheuristic technique to approximate the global optimization in an extensive search area. As can be seen in [Kir84], it is inspired by the annealing process in metallurgy.

- Mathematical programming [Min86] is a family of optimization techniques to maximize or minimize a function, systematically choosing input values among some permitted values and calculating the value of the function. This function can be subject to certain restrictions expressed in terms of equalities and/or inequalities. Depending on the type of functions and restrictions, as well as on the values assigned to them, there are different variants of mathematical programming.

  The objectives of Linear Programming (LP) [Sch98], which forms part of mathematical programming, is to maximize or minimize a linear function subject to restrictions, which is formulated through simple linear equalities or inequalities. If all the variables are integer numbers, it is denominated Integer Linear Programming (ILP), while if only some are, it is denominated Mixed Integer Linear Programming (MILP). If all equalities and inequalities are non-linear, it is simply denominated Mixed Integer Programming (MIP). Geometric Programming (GP) [Boy+07] is the type of mathematical programming used to maximize or minimize polynomial functions with restrictions. These are formulated through monomial equalities equal to 1 and posynomial inequalities less than or equal to 1.

  Constraint Satisfaction Programming (CSP) is a type of mathematical programming based on determining the value of the variables guaranteeing fulfillment of an objective function [Tsa14]. If the variables of the objective function and the functions of restrictions are Boolean, it is called a Boolean Satisfaction Problem or simply SAT. Lastly techniques based on Satisfiability Modulo Theories (SMT) are also considered, which consist of formulas in first-order logic in which some function has additional interpretations, and it must be determined whether a formula can be satisfied [BT18].

- Branch and Bound (BB) [LD60] is a generic algorithm used in combinatorial and discrete optimization problems and mathematical optimization. The candidate solutions are numbered forming a tree and discards are made based on estimations above and below the factors that must be optimized.

- A heuristic (HEU) [Pea84] is a problem-solving method founded on rules built according to criteria linked to the problems themselves, generally based on previous experience. Although finding an optimal solution is not guaranteed, reasonably good solutions can be found in an acceptable time if they are well constructed.

The review of the works is done according to the following methodology. On the one hand, the problems addressed in each one are reported, from the viewpoint of allocation, type of scheduling and partitioning. These characteristics are fundamental in those real-time systems that implement functionalities with different levels of criticality. The review will also focus on the objectives of each work. They all share the aim of system schedulability, and we will also consider those aiming to minimize the following parameters: the number of computers, resource utilization and response times. Lastly, the restrictions presented in each work in terms of the use of memory and the relation between the deadlines and the periods (D/T) in the system model described are identified, where possible. All these characteristics are shown in tables as a summary, which constitutes a guide to finding in an easy way the solutions available for a specific problem of interest.

2.2 Genetic Algorithm (GA)

In [MR93], [HAR94] and [MBD98] different variations of GAs are used to allocate tasks in processing elements. All of them are used to create cyclic schedulings with the aim of minimizing their response times. In [MR93] and [MBD98] e2e flows defined through their deadlines and periods are considered, and in [HAR94] a multiprocessor system is proposed.

In [DJ98] a multi-objective algorithm is developed to decide on the amount of hardware resources in a System On Chip (SoC), as well as to allocate the tasks in the processor nodes and to build cyclic executives. The combined use of GA and TS aims to minimize energy consumption and the price of the resultant system, while meeting all the temporal restrictions.

The work [FDB00] also proposes the allocation of periodic tasks and the creation of cyclic scheduling. However, the system model considers identical processors, connected through Time Division Multiple Access (TDMA hereafter) networks. To do so, turns are explicitly assigned.

In the works [OW04] and [Yoo09], GAs are used to allocate tasks in identical processors and to determine cyclic scheduling in distributed real-time systems. Both consider e2e flows and predefined deadlines, and their principal objective is to minimize the sum of all the differences between the worst-case response time and the deadline of all the flows. They also aim to minimize the number of processing elements. These works are differentiated by the convergence mechanisms of the GA towards a suitable solution.

In [Ham+06], a multi-objective GA is proposed to assign priorities to tasks, as well as to determine the timing slots for the messages transmitted through a TDMA network. A specific timing analysis technique is used, which enables the adjustment of the variables and objectives to be minimized.

In [SDJ07], a system model is considered, based on computers with memory resources and e2e flows whose period and deadline are predefined. Its main objective is to minimize price and energy consumption of the resultant system, in which the number of processors, communication elements and FPGAs are determined. Scheduling is carried out through a heuristic algorithm that uses Pareto optimization criteria to evaluate the candidate solutions [GH88], [FF98].

In [Sam+09], genetic algorithms are used to assign priorities to tasks and fixed priorities and frame identifiers to messages in a distributed real-time system connected through a FlexRay network. The main aim is to optimize the average response time of the system.

The work [Azk+11b] develops a permutational coding for a GA whose objective is the assignment of priorities to tasks and messages making up a distributed real-time system. The results obtained in this work show an improvement on those obtained by [GG95] using a heuristic called HOPA. Later, in [Azk+12], this same coding is used to schedule the tasks of that system, this time using HOPA to generate the first population of solutions that the algorithm will optimize. Finally, in [Azk+11a], a GA is used to allocate and schedule tasks and messages in distributed real-time systems, minimizing parameters such as utilization of computers, memory resources and communications, response times and the number of computers used, while meeting all the real-time requirements.

In [Woz+13], a GA is used for the synthesis of distributed real-time systems, composed of heterogeneous processing elements. The synthesis process is made up of the following phases: allocation of software components (composed of sets of runnables) in processing elements and signals from the communications buses, decomposing the runnables into tasks and signals into messages, and assignment of fixed priorities. As well as meeting all the deadlines, it has the aim of optimizing some parameters: the response time of the e2e flows, the use of memory, the performance of the communications buses and the response time of the runnables.

The work [BO14] proposes a GA to minimize the response time of a real-time system with periodic and aperiodic tasks, while maintaining an equilibrated use of the processor elements. It considers flows with pseudo-periods and also the use of fixed and dynamic priorities. It uses a correcting technique in the solutions to guide the initialization of the algorithm and to maintain the precedence relations of the tasks. In [BO16], a stage based on quantum logic coding is added to the genetic algorithm.

In [Aya+16a], a genetic algorithm is used to allocate independent tasks in heterogeneous distributed real-time systems, to which fixed priorities are assigned following the Rate Monotonic scheme. The total execution time, the cost of communications among tasks and memory consumption are expressed in suitable fitness functions that must be minimized. In the next work [Aya+18], an advanced genetic algorithm (ImGA) is proposed that implements a crossover operator based on schedulability of tasks [Aya+16b], which improves the precision of the genetic algorithms previously used.

In [WME20], the authors develop a genetic algorithm to generate feasible schedules in hierarchically scheduled multicore systems, by means of novel chromosome coding and fitness functions.

After reviewing the works using genetic algorithms, it can be highlighted that this approach is used habitually as a multi-objective optimizer, given that they search for solutions in which more than one parameter has to be minimized (costs, number of computers, etc.). The reviewed works are collected in Table 2.1.

Tab. 2.1: Reviewed Works - Genetic Algorithm

Work	Addressed Problems			Minimization Objective			Restrictions	
	Allocation	Scheduling	Partitions	Computers	Utilization	Timing	D/T	Memory
[MR93]	Yes	Cyclic				Response		
[HAR94]	Yes	Cyclic				Response		
[MBD98]	Yes	Cyclic				Response	$D = T$	
[DJ98]	Yes	Cyclic		Cost	Energy		$D < T$	
[FDB00]	Yes	Cyclic					$D = T$	
[OW04]	Yes	Cyclic		Number		Response		
[Yoo09]	Yes	Cyclic		Number		Response		
[Ham+06]		Priorities					$D < T$	
[SDJ07]	Yes	Cyclic		Cost	Energy			Yes
[Sam+09]		Priorities				Response		
[Azk+11a]	Yes	Priorities		Number		Response		
[Woz+13]	Yes	Priorities				Response		Yes
[BO14]	Yes	Priorities				Response		
[Aya+16a]	Yes	Priorities				Response		Yes
[WME20]	Yes	Hierarchical	Yes			Response	D=T	

2.3 Tabu Search (TS)

In [PKR00], the use of TS is proposed to allocate tasks with precedence relations in processors and create cyclic scheduling for real-time, heterogeneous multiprocessor systems, composed of identical processor except for one with higher processing power. The principal objective is to minimize the system's response time.

In [CL00], a hybrid algorithm is developed based on TS to solve the problem of allocating tasks with precedence relations in a multiprocessor architecture. The objective is to meet the real-time requirements of the system while satisfying the restrictions in maximal computation capacity of the computers, as well as to minimize the network traffic and the number of computers utilized.

The work [LKY00] proposes the values of the operators and configuration parameters of the TS and GA algorithms, with the objective of allocating real-time tasks and creating cyclic scheduling in multiprocessor architectures, whose processors are considered identical. The objective is to minimize the system's global response time.

In [TP11b], an optimization method is proposed for partitioned distributed real-time systems. Through an algorithm based on TS, it attempts to allocate tasks in processing elements and to define their temporal execution windows within the MAF. The algorithm is also in charge of assigning tasks to these temporal partitions and generating a cyclic executive. The objective is to meet the deadlines of all the tasks at the same time as maximizing the downtime of the temporal partitions. Later, Tab. 2.2: Reviewed Works - Tabu Search

Work	Addressed Problems			Minimization Objective			Restrictions	
	Allocation	Scheduling	Partitions	Computers	Utilization	Timing	D/T	Memory
[PKR00]	Yes	Cyclic				Response		
[CL00]	Yes	Priorities		Number	Net. & CPU	Response		
[LKY00]	Yes	Cyclic				Response		
[JPJ17]		Priorities			Energy	Response	$D < T$	
[TP11b]	Yes	Hierarchical	Yes			Response	$D < T$	
[TP15]	Yes	Hierarchical	Yes		Cert. cost	Response	$D < T$	

the work is extended in [TP15], where a series of techniques is proposed to enable the scheduling of the applications, for which the costs of certification are taken into account to host mixed criticality applications, which are minimized using different techniques.

In [JPJ17], a multiobjective algorithm based on TS is used to schedule tasks and messages in embedded distributed real-time systems. The objective is to develop an implementation that meets the system's real-time requirements, as well as the safety and dependability ones, while minimizing the energy consumption. Priorities are assigned to the tasks in descending order, and it is assumed they have been allocated in the processors previously.

Although there are not many works in which TS is used, it can be seen in Table 2.2 that varied problems and optimizations are tackled, including partition-based systems.

2.4 Simulated Annealing (SA)

In [TBW92], SA is used to allocate tasks and to assign them fixed priorities in distributed real-time systems, in which a TDMA communications network is used. In this system model, memory resources of the computers are considered and the tasks have a series of candidate computers in which they can be allocated. Moreover, some tasks are replicas of others and cannot be allocated in the same processor. The deadline of the tasks is equal to their period, and the priorities are assigned following the Rate Monotonic criterion. The objective of this work is to minimize the network load while all the tasks meet their deadlines. Another restriction to be considered is the use of the memory of the computers, which must not be surpassed.

The work [Bur+93] proposes an algorithm based on SA to allocate tasks in processing elements and to assign them fixed priorities. It is considered that each task has a series of candidate computers in which it can be allocated. It also considers e2e flows with periods and deadlines. The objective is to minimize the worst-case response time of all the flows as well as the number of auxiliary tasks that route communications.

In [CP95], SA is used to generate cyclic schedules in distributed real-time systems. The objective is to minimize the jitter of the tasks while meeting the deadlines of the e2e flows.

In [DS95], SA is used to create a cyclic scheduling in distributed real-time systems based on flows made up of periodic tasks. The objective is to minimize the jitter of the tasks while the deadlines of the tasks and flows are met.

The work [VO05] presents a comparative study of Multi-start [Mar03], SA and TS algorithms used for scheduling distributed real-time systems, connected through a *Foundation Fieldbus* [VP08] communications network. The results show that SA is the most difficult algorithm to configure and that TS obtains more numerous and better results.

In [HGZ10], SA is used to allocate tasks in homogeneous computers, as well as to assign priorities and periods to the tasks and to configure access to the communications network based on Time-Triggered Protocol (TTP hereafter) [KG93b]. The deadlines of the tasks are equal to their periods, and the objective is to minimize the worst-case execution time of all the e2e flows. The proposal is based on two phases. The first uses SA for task allocation. Later, on top of this solution, GP is used to fix the deadlines of the tasks and to manage access to the communications network.

In [EB10], the use of SA is proposed for the allocation of tasks and messages and assignment of priorities with the aim of facilitating the extension or updating of scenarios, minimizing the impact of these changes in terms of the requirements of the original system.

In [TP11a], the use of SA is proposed to solve a problem of optimization of distributed real-time systems with partitions. The temporal partitioning is implemented through time slices in which the execution of tasks is divided within a processing element, and it is the algorithm that decides on the order and length of these portions within the processor. It is assumed that the tasks are previously allocated in the processors. The scheduling is cyclic for the critical tasks, while the non-critical ones are scheduled using preemptive fixed priorities. The objective is to optimize the sequence and length of the temporal windows and to obtain schedules that permit all the safety-critical applications to meet their deadlines.

Tab. 2.3: Reviewed Works - Simulated Annealing

Work	Addressed Problems			Minimization Objective			Restrictions	
	Allocation	Scheduling	Partitions	Computers	Utilization	Timing	D/T	Memory
[TBW92]	Yes	Cyclic			Network	Response	$T = D$	Yes
[Bur+93]	Yes	Cyclic			Routing tasks	Response		
[CP95]	Yes	Cyclic				Response		
[DS95]		Cyclic				Jitter		
[VO05]	Yes	Cyclic						
[HGZ10]	Yes	Priorities				Response	$D = T$	
[TP11a]	Yes	Hierarchical	Yes			Response	$D \leq T$	
[PA15]	Yes	Priorities				Response	$D = T$	
[McL+20]	Yes	Priorities				Response	$D \leq T$	

In [PA15] a comparative study is also made, this time between SA and GA. The scheduling of tasks is evaluated in homogeneous multiprocessor architectures, with the objective of minimizing the response time of the system respecting the precedence relations of the tasks. The study demonstrates that after 9 case studies, SA obtains 6 acceptable solutions while GA obtains 5. Most of the works that made use of SA have approached the allocation problem as well as the scheduling one. The minimization of the jitter of the tasks is addressed for the first time.

In [McL+20], the authors use SA to address the allocation and scheduling of autonomous driving applications on multicore systems. They propose several strategies such as deadline and offset adjustments and random task re-allocations, and their results considerably improve previous works based on heuristics and GA.

Most of the works that use SA have addressed the task scheduling and allocation problems combined. As shown in Table 2.3, the minimization of tasks' jitter is addressed for the first time.

2.5 Mathematical Programming

In [SK01], CSP is used to allocate tasks in processing elements and to create cyclic scheduling in distributed real-time architectures. These architectures are composed of heterogeneous networks that connect computers to memory resources and Application Specific Integrated Circuits (ASIC). The tasks are selected sequentially and allocated in the processor with the lowest cost. After the allocation, the BB technique is used to assign an execution time to each task. The objective is to minimize the time of all the scheduling without overusing the memory resources. In [SK03], a method is presented to facilitate the allocation problem and the scheduling one previously mentioned.

In [EJ01], CSP is used to allocate tasks in computers and to create cyclic scheduling in distributed real-time systems. The computers are heterogeneous and they are connected through a communications bus. The principal contribution of this work is the evaluation of several local search heuristics that decide which variables and what values are assigned in each iteration. There are 3 objectives: minimize the total time of the scheduling, minimize the communications through the bus and maintain an equilibrated use of the processor elements.

In [MH06], SAT is used to allocate and schedule tasks in distributed real-time systems composed of heterogeneous processor elements. For this, priorities are assigned to the tasks of each computer following the Deadline Monotonic scheme. Each task has a period, a deadline, memory requirements and a subset of candidate computers in which to be allocated, and some of them that cannot be allocated in the same computer are specified as another subset.

In [Dav+07], GP is used to optimize the periods of tasks and messages. In this case, they are already allocated and the priorities have also been previously assigned, so the objective is to minimize the sum of the worst-case execution times.

In [Zhe+07b], MILP is used to allocate tasks in computers, package signals into messages and assign fixed priorities to the tasks and messages in the distributed real-time systems based on CAN networks. This model considers e2e flows composed of tasks that exchange messages and which have deadlines and periods. The objective is to minimize the worst-case response time of the flows while their deadlines are met. Given that on industrial scale the techniques based on MILP are computationally very costly, the problem is divided into two sub-problems that can be solved separately.

Strictly periodic applications can have much longer response times than those activated by events. Therefore, when a deadline of a task or message is not met, it is possible to apply the ET paradigm so its response time decreases and the deadline can be met. In [Zhe+07a], MILP is used to adjust the activation model with the objective of meeting all the temporal restrictions.

In [Hla+08], CSP is used to allocate and schedule tasks in distributed real-time systems composed of homogeneous computers connected through a CAN network. It is assumed that the tasks and messages have deadlines equal to their periods, and that the priorities are assigned in a phase previous to this process. All the tasks have a subset of candidate computers in which they can be allocated, and they can be required to be allocated in the same or a different computer than other tasks. The objective is to obtain an allocation enabling all the deadlines to be met. To do so, two methods are proposed. The first is based on backtracking and it simultaneously tackles the problems of allocation and scheduling. The second method is to divide these two sub-problems according to a Benders scheme [Sch98] to tackle them separately.

In [Zhu+09], MILP is used to allocate tasks in processors and signals in messages, as well as to assign fixed priorities to tasks and messages, in distributed real-time systems that assume a model based on e2e flows with fixed deadlines and harmonic periods. The objective is to maximize the extensibility: to prolong as much as possible the worst-case execution time without affecting the fulfillment of the temporal restrictions of the tasks while ensuring that all the deadlines are met. The method proposed assumes an initial assignment of priorities for messages and tasks.

In [ABH10], an algorithm based on MILP is proposed, used to allocate and schedule strictly periodic temporal partitions in distributed architectures based on Integrated Modular Avionics [Joh99]. A series of restrictions is defined mathematically (avoiding temporal overlapping, shared access to resources, communication costs, etc.) and solved through a commercial tool, obtaining nearly optimal results.

In [Zhu+12], MILP is used to allocate tasks on distributed architectures with heterogeneous processors interconnected through CAN networks, which are at the same time segmented through gateways. Priorities are also assigned to the tasks and messages, with the aim of meeting hard real-time requirements and minimizing the latency of the e2e flows.

In [CO14], CSP is proposed to allocate and schedule tasks and messages in distributed real-time systems. After building the logic restrictions that define the system, it uses two algorithms based on SMT. The first, called One-shot, tackles the scheduling of all types of tasks defined in the system model. The second method does not consider the scheduling of the independent tasks at a first approach, in such a way that no difficulty is added to the formulation of the SMT algorithm. After carrying out the scheduling based on the algorithm proposed in [COE14], the independent tasks are added. This work considers the scheduling of the processor elements and of the communications network, and in the following work [Cra+16], the schedulability analysis is applied to TSN networks.

In [Zha+14], MIP is used to schedule distributed real-time systems, at both computer and network level. It is assumed that the tasks are allocated in the computers with a single processor. The objective, represented through the objective function, is to minimize the response times of the applications that are executed while meeting all the deadlines of the tasks. The temporal restrictions are represented through equalities and inequalities, for which the maximum tolerable response time can be established.

In [Alt+12] and [Alt+14], an algorithm is presented based on ILP formulations that are used for allocating and scheduling of periodic tasks in distributed real-time systems. The architecture is based on multiple subsystems that are communicated through a global bus. The tasks are scheduled following the Deadline Monotonic scheme and they form e2e flows along with the messages they use to communicate.

In [Min+18], three approaches are proposed to schedule tasks and messages with jitter and precedence relations in real-time systems. It is assumed that the allocation of the tasks in the computers has already been done. First, an approach based on SMT is formulated, in which the solutions space is defined through five sets of restrictions. Later, an ILP model is described in a similar way, with the difference of the linear restrictions this method needs. Although the solutions obtained in these two approaches are optimal, a third heuristic based on three levels is proposed, which although not optimal, does obtain acceptable solutions in a reasonable time and with a much greater scalability of the problem.

In [Bli+18], an algorithm based on MILP is developed to schedule periodic tasks in distributed real-time systems based on IMA architecture. Through mathematical formulation of the real-time restrictions and requirements of the system, a cyclic executive is constructed whose period is the minimum common multiple of the periods of the tasks. It is assumed that the tasks have already been allocated. It uses partitions in the following way: the nodes forming this architecture implement different modules to provide the spatial partitioning, and through the cyclic executive resulting from the scheduling, temporal isolation is obtained. Within each temporal partition, the tasks are scheduled according to the RMS scheme.

Finally, [Gua+20] proposes an ILP formulation for scheduling real-time tasks in uniprocessor systems. The authors state that this approach can be applied to hierarchical schedulers, although they do not address distributed architectures.

As shown in Table 2.4, this method is one of the most widely used, but in general, several problems are not tackled at the same time, maybe due to the complexity of their mathematical formulation. When there is no specific minimization objective, these mathematical formulations are used to obtain a schedulable solution.

Tab. 2.4: Reviewed Works - Mathematical Programming

Algorithm	Work	Addressed Problems			Minimization Objective			Restrictions	
		Allocation	Scheduling	Partitions	Computers	Utilization	Timing	D/T	Memory
CSP	[SK01]	Yes	Cyclic			Memory	Response		Yes
	[EJ01]	Yes	Cyclic			Network / Processor	Response	$D = T$	
SAT	[Hla+08]	Yes	Priorities					$D = T$	
	[MH06]	Yes	Priorities						
GP	[Dav+07]		Priorities				Response		
MILP	[Zhe+07b]	Yes	Priorities				Response		
	[Zhe+07a]		Priorities						
	[Zhu+09]	Yes	Priorities					Harm. T	
	[ABH10]		Cyclic						
	[Zhu+12]	Yes	Priorities				Response		
	[Bli+18]	Yes	Hierarchical	Yes					
CSP	[CO14]	Yes	Cyclic					$D \leq T$	
MIP	[Zha+14]		Cyclic						
ILP	[Alt+14]	Yes	Cyclic					$D \leq T$	
	[Gua+20]	Yes	Cyclic						
SMT - ILP	[Min+18]		Cyclic						

2.6 Branch and Bound (BB)

In [HS97], BB is used to allocate tasks in computers and generate cyclic schedules in distributed real-time systems composed of identical computers. The objective is to maximize the probability that all the tasks meet their deadlines.

In [PSA97], BB is used to allocate tasks in computers and create cyclic scheduling in distributed real-time systems composed of heterogeneous computers. A model of e2e flows is assumed with a period and a deadline, composed of tasks with precedence relations and which can communicate with the tasks of other flows. Moreover, the tasks can have different worst-case execution times in different computers. The proposed algorithm allocates all the tasks of a flow in a computer and establishes their execution order, and it has the aim of minimizing the ratio of worst-case response time and the deadline of the flows.

In [RRC03], tasks are allocated in processor elements and they are assigned fixed priorities using BB. The architecture is composed of different sets of computers connected through a CAN communications network. Within each one of these sets, all the computers are identical, although among different sets they can be different. The tasks have specific deadlines and they can communicate among themselves. The scheduling is done in each subset of computers. The tasks are allocated following an increasing order of deadline, and they are assigned priorities following the DMS scheme. The branches of the search tree are pruned depending on a lower limit of worst-case response time.

BB is one of the least widely used methods, given that as can be seen in the few available works collected in Table 2.5, they are from a long time ago in comparison

Tab. 2.5: Reviewed Works - Branch and Bound

Work	Addressed Problems			Minimization Objective			Restrictions	
	Allocation	Scheduling	Partitions	Computers	Utilization	Timing	D/T	Memory
[HS97]	Yes	Cyclic						
[PSA97]	Yes	Cyclic				Response		
[RRC03]	Yes	Priorities						

with other works included here. With this method, both cyclic and priority-based scheduling have been tackled.

2.7 Heuristics (HEU)

In [GG95], the heuristic denominated Heuristic Optimized Priority Assignment (HOPA) is developed for the assignment of fixed priorities to tasks and messages in distributed real-time systems. First, individual deadlines are assigned to the tasks and messages making up the e2e flows, and then the DMS scheme is used to assign the priorities. The worst-case response times are calculated through a holistic analysis or through timing analysis based on priorities. If some deadline is not met in any flow, some metrics are calculated to determine the deviations of that flow over its deadline and the individual deadlines are calculated again, and the priorities are assigned. This process is repeated until a valid solution is obtained or a maximum number of iterations is reached.

In [Ram95], a heuristic is proposed to allocate tasks in homogeneous computers and create a cyclic scheduling. It is based on a model composed of deadlines and periods, and it considers the possibility of allocating some replicated tasks in different computers. The first phase of the algorithm determines the suitability of grouping and allocating pairs of tasks that communicate among themselves in the same computer. The second allocates and schedules the tasks following the Latest Start Time / Maximum Immediate Successors First (LST/MISF) criterion.

In [Bra+01], a comparative study is done on eleven heuristics used to allocate tasks and create static scheduling. The model is composed of independent tasks with different worst-case execution times depending on the processor element in which they are allocated, and the objective is to minimize the response time of the total scheduling. Among the eleven, the best results are obtained by a genetic algorithm and a heuristic that first allocates the tasks with the shortest execution time in the computer in which they have the shortest worst-case execution time.

The same research group presents in [Bra+08] another comparative study of three methods to allocate tasks in computers and to generate cyclic scheduling in distributed real-time systems composed of heterogeneous computers. The tasks have precedence relations, priorities, deadlines and more than one version of each one of them. They also have different worst-case execution times depending on the computer they are allocated in, and the longer the execution time, the greater the priority they are assigned. The algorithms compared are a heuristic and two genetics. The heuristic obtains good results, and the two genetics improve on these results by 2% and 5% respectively.

In [Ali+02], three heuristics are proposed for assignment of resources to e2e flows in distributed real-time systems composed of computers based on Round Robin (RR) scheduling. The e2e flows are considered with deadlines and periods. The system load is modeled through the rate of generation of information from the input sensors to the e2e flows. The objective is to allocate the tasks so as to maximize the permitted growth of load.

In [Ele+00], heuristics are proposed to allocate tasks in computers and create cyclic schedules of tasks and messages in distributed real-time systems composed of different types of processors. A scheduler based on a list-scheduling heuristic is responsible for assigning the resources to the tasks or messages that are activated depending on their priorities. Likewise, a heuristic is also developed to optimize the scheme of access to a TTP network. The same research group has tackled the optimization of the TTP access scheme in [PEP00] and [PEP04b], where they propose and analyze different messaging strategies and they present heuristics to optimize each one of them.

The works [PEP03b] and [Pop+04] propose heuristics to allocate tasks in computers and schedule tasks and messages in distributed real-time systems that combine the TT and ET paradigms. The model is based on a group of TT type computers connected through a TTP communications network, another group of ET type computers connected through a CAN network and a gateway that links the two networks, and through which messages are exchanged between the two groups. The heuristics proposed allocating the tasks in one of the two groups, allocating the tasks in one of the computers of the group and scheduling the tasks and messages in their respective resources so that the deadlines of the flows are met.

Another approach by the same authors in [Pop+01a] [Pop+01b] is the allocation and scheduling of applications in distributed real-time systems with the aim of minimizing changes in the applications already allocated in the system, and the maximization of the probability that future applications can be allocated in the same system. To do so, some metrics are proposed that model the probability of future applications requiring a certain processing time in each particular period. In [PEP03a], [Pop+05] and [Pop07], methods are proposed to generate and optimize the scheduling of mixed TT/ET systems. One of the methods is based on ILP and is optimal, while the other is a heuristic that obtains sub-optimal results in shorter times. The heuristic is guided by some rules whose objective is to increase the degree of scheduling of the system through modification of the scheduling policies of the tasks, the allocation of tasks in the computers and the access scheme to the communications network.

In [PA04], two heuristics are proposed for the assignment of local deadlines to messages with global deadlines that cross through multiple TT networks from their origin to their destination. The first heuristic is denominated Isometric Allocation (ISO) and is based on dividing the global deadline into as many equal parts as the message passes through and assigning in each one a local deadline equal to those parts. The second heuristic is Maximum Schedulability Laxity (MSL) and it is based on assigning local deadlines in a way proportional to the bandwidth available in each one of the networks that are crossed.

In [QJ06], an algorithm is presented for the allocation and assignment of priorities to tasks in distributed real-time systems. The processors are heterogeneous, so the tasks will have different execution times depending on which one they are in. All tasks have a copy that is executed in the case of a fault and which must be allocated in a different computer. The objectives are to minimize the response time of the system in such a way that the deadlines are met, as well as extending the reliability of the system as far as possible.

In [Pop07], a series of heuristics is proposed to synthesize, analyze and optimize the allocation of tasks in computers, the assignment of priorities to tasks, and the assignment of communication slots to computers in distributed real-time systems. The combination of the TT and ET paradigms is done both at computer level and at communications network level. The TT tasks are activated in predefined instants in scheduling tables, have maximum priority and cannot be discarded. As for the ET tasks, they are activated through external events and messages, and are preempted by the TT tasks and the ET tasks of higher priority. All the tasks of a flow are of one of the two types described. Given that the combination of tasks and schedulers based on different paradigms requires the extension of the existing schedulability analysis techniques, a heuristic scheduling and an extension to the holistic analysis [TC94] for systems combining TT and ET tasks are proposed.

Following the system model described in [ABH10], in [Al +11], a heuristic based on game-theory is used to allocate and schedule temporal partitions in distributed architectures based on IMA. This algorithm works with the concept of players who adapt their strategies based on the most recently observed strategies of the other players. Nearly optimal results are demonstrated in a much shorter computation time than that achieved by [ABH10]. The objective, as well as to obtain a scheduling that enables meeting the real-time requirements, is to maximize the idle time of the processor within the MAF to be able to extend the system functionalities in the future without affecting the scheduling.

In [Eis+10], a hybrid method based on heuristics and mathematical formulations is proposed to solve the allocation and scheduling of periodic tasks in distributed real-time systems. The architecture corresponds to that of an airplane, and it is divided into two cabinets, formed by identical processor elements communicated through a communications network. The objective is to minimize the number of processors while all the deadlines of the tasks are met. These tasks can have requirements of cohabitation or mutual exclusion. The scheduling of the communications network is not considered.

The work [NSE11] develops a heuristic for the assignment of fixed priorities to tasks and messages in distributed real-time systems based on a model of flows with periods and deadlines. For each schedulable resource, a priority assignment module is executed along with a commercial temporal analysis module. This heuristic obtains poorer results than the genetic algorithm proposed in [Ham+06], although it requires much shorter computation times.

In [Meh+13], the previously mentioned work [Woz+13] is extended, considering the same system model. The objective is to minimize the latencies of the e2e flows while meeting the hard real-time requirements. A heuristic is developed based on two stages. First, the problem of distributing the runnables in tasks is tackled and then the one of their allocation and scheduling is faced. The second stage is based on an iterative two-level process which provides as a result an optimal solution, although only for very small-scale systems.

In [Klo+13], a method is presented for the allocation and scheduling of tasks and messages in distributed real-time systems based on AUTOSAR. The objective is to obtain a reconfigurable ECU-network topology that is fault tolerant, while meeting the real-time requirements. When faults are detected, redundant tasks in different nodes are activated. The scheduling of each computer is done by priorities according to DMS.

In [Gar+14], an algorithm is shown that is used to schedule parallel and distributed tasks with real-time requirements. The so-called fork/join tasks are scheduled using fixed priorities according to the DMS scheme and they are executed in a distributed platform composed of processor elements of one single identical core, interconnected through a real-time network. In a later work [Gar+15], a heuristic called DOPA (Distributed using Optimal Priority Assignment) is proposed, as an extension of the classic OPA [Aud91] applied to distributed real-time systems. This algorithm enables two interrelated problems to be tackled. First, the optimal assignment of priorities is done for independent tasks, for which the tasks with precedence relations must be transformed into independent ones through intermediate deadlines. The second problem is related to the decomposition of executables in schedulable tasks. The algorithm also attempts to accommodate the greatest number of tasks within the same computer to minimize the message load sent through the network.

In [YR15], an algorithm is developed for scheduling tasks in distributed real-time systems based on AUTOSAR architectures. It consists of decomposing the e2e flows in tasks with local deadlines and assigning them priorities, in such a way that these local deadlines are met. First, the algorithm solves the two problems separately, and then another algorithm is developed that tackles them together. No type of scheduling in the communications network is considered.

In [Hu+15], an algorithm is proposed to schedule tasks and messages in distributed real-time systems, in which the allocation of tasks on computers is assumed to be done previously. The e2e flows can be composed of tasks and periodic or aperiodic messages, and the objective is that all the instances of all the applications meet their deadlines. The algorithm first orders the flows assigning them priorities, and then the tasks and messages that form them are scheduled according to the SHLF algorithm (Synchronized Highest Level First). Finally, two procedures are detailed to provide support to future re-scheduling.

In [CDH16], time/space partitions are modeled as strictly periodic, non-preemptive tasks, which are allocated and scheduled on distributed architectures based on IMA. First an algorithm based on MILP is applied to obtain a maximum scaling factor to apply to each temporal partition, and then a heuristic based on game theory enables the determination of the allocation and scheduling of each one of them. Real-time requirements can be met adjusting the scaling factor to the specific temporal requirements.

In [Xie+16], tasks are allocated and scheduled in a distributed real-time system, composed of heterogeneous computers and several networks interconnected through gateways that implement mixed criticality functions. The scheduling is done through

RR, and the objective of the algorithm is to minimize the ratio of unsurpassed deadlines while maintaining satisfactory performance.

In [DSF16], an algorithm is developed for the allocation and scheduling of partitions in the processors of distributed real-time systems. The temporal partitions are previously defined through their periods and their deadlines. They are grouped in flows that are communicated through messages that are also considered for the computation of total worst-case response time. The algorithm itself decides the number of processors to be used. In [DSF17], this work is extended using an algorithm that limits the search space of the optimal solution, discarding possible sub-optimal solutions in each stage of the algorithm.

In [BSR17], an algorithm is developed to allocate tasks in processing elements of a distributed architecture, scheduled according to the RMS scheme. The objective of the algorithm is to minimize the number of processors and the latencies of the tasks allocated in them, while meeting all their deadlines. It is assumed that these deadlines are equal to the period of each task.

In [Zho+19], the authors propose a method for scheduling partition-based real-time systems. They propose a task-to-partition mapping strategy and generate TDMA-like partition schedulers, taking into account context switch overheads.

The work [ZZ19] proposes an iterative algorithm for fixed priority assignment in both preemptive and non-preemptive scheduling policies. However, it does not consider time-partitioned architectures like the one addressed in this work.

This algorithmic approach is the most widely used in the optimization of the allocation and scheduling in distributed real-time systems. This is probably because they are efficient algorithms, designed ad-hoc to solve the problem being tackled.

2.8 Classification of works and conclusions

Tables 2.1 to 2.6 compile all the works reviewed in this chapter. They show the most characteristic aspects of each contribution according to the methodology explained and the detailed review just made. It should be highlighted that in all the works compiled the objective is the schedulability of the system, although this is not explicitly shown in the table.

The allocation and scheduling of distributed real-time systems have been tackled in a large number of works over time. There is great diversity in the approaches

Tab. 2.6: Reviewed Works - Heuristics

Work	Addressed Problems			Minimization Objective			Restrictions	
	Allocation	**Scheduling**	**Partitions**	**Computers**	**Utilization**	**Timing**	**D/T**	**Memory**
[GG95]		Priorities				Response		
[Ram95]	Yes	Cyclic					$D = T$	
[Bra+01]	Yes	Cyclic				Response		
[Bra+08]	Yes	Cyclic				Response		
[Ali+02]	Yes	RR						
[PA04]	Yes	Priorities / Cyclic					$D = T$	
[Ele+00]	Yes	Cyclic				Response		
[PEP00]	Yes	Cyclic				Response		
[Pop+01a]		Priorities					$D \leq T$	
[Pop+04]	Yes	Cyclic					$D \leq T$	
[Pop07]	Yes	Priorities / Cyclic						
[QJ06]	Yes	Priorities				Response		
[NSE11]		Priorities				Response		
[Eis+10]		Cyclic		Number			Harmonic T	
[Al +11]	Yes	Cyclic	Yes			Response		
[Meh+13]	Yes	Priorities				Response		
[Klo+13]	Yes	Priorities				Response		
[YR15]	Yes	Priorities				Response		
[Gar+15]	Yes	Priorities			Network Use		$D \leq T$	
[Hu+15]		Priorities						
[Xie+16]	Yes	RR						
[CDH16]	Yes	Cyclic	Yes					
[DSF16]	Yes	Cyclic	Yes					
[BSR17]	Yes	Cyclic		Number		Response	$D = T$	
[Zho+19]	Yes	Hierarchical	Yes			Response	$D \leq T$	
[ZZ19]	No	Priorities				Response	$D \leq T$	

used when proposing optimization algorithms for these systems; although all of those included in this work have the common objective of meeting their real-time requirements, there are some that simply propose the assignment of priorities to tasks [GG95], while others propose the elaboration of cyclic executives [DS95]. Others such as [HGZ10] or [Pop07] tackle as well as the scheduling, the allocation of the tasks and the messages in specific architectures. Moreover, while maintaining the main objective of meeting all the deadlines, some works attempt to optimize certain parameters such as the use of resources, either of computation or equipment [Azk+11a].

This review does not have the objective of carrying out a comparative study among the works included here. Clearly, to be able to carry out a comparison to determine which of all the works obtains the best results, the system models of all of them should be completely homogeneous, as should the case studies on which they have been implemented and validated. This would be impossible, so the appreciations that can be extracted from the results obtained in this study are qualitative comments obtained through an overall interpretation of the table. This in itself constitutes a result that can be used by future researchers to find the references that are most suitable for the models or algorithms with which they wish to work.

It should be remarked, as can be seen in the tables, that there is a small number of works tackling allocation and scheduling of distributed real-time systems based on partitions. This confirms that there is an open path for research into techniques that consider this characteristic, so necessary in the development of the most modern mixed-criticality applications. The advances in the research on timing analysis techniques for partitioned systems will undoubtedly help to increase the number of solutions proposed for this relevant problem.

Another remarkable aspect is that the vast majority of the works search for configurations in which the system's response time is minimized. This is coherent; given that in systems where not meeting deadlines can produce serious consequences, minimizing the response time reduces the probability of this occurring. Lastly, it could be said that after exhaustive analysis of the state-of-the-art, there are realistic system models that have still not been studied. As an example, the combination of multipath e2e flows with hierarchical schedulers has never been addressed. It has also been found that specific algorithms and techniques for searching and optimizing have not been applied on systems that follow the model and the scheduling schemes used in this thesis. For instance, Simulated Annealing approaches have recently provided promising results and future research might consider this approach.

Real-time system model 3

In this chapter a complete description of the system model considered in this thesis is presented. It is compliant with MAST [Gon+01] and also with the second version of its metamodel, MAST2 [Har+13]. The terminology used throughout this model's description is aligned with OMG's MARTE standard [Obj11]. After having presented the system model, the railway signaling application described in chapter 1.2 is modeled according to this description, which will be used in the rest of the thesis in order to develop the analysis and optimization algorithms.

3.1 Logical architecture

The logical architecture is composed of distributed *end-to-end flows*. They contain a kind of event handler called *step*, which represents an operation being executed by a schedulable resource (a task or message) in a processing resource (a processor or network) with certain scheduling parameters. Steps are activated from an input event and generate an output event when they finish their execution. Formally, the i-th e2e flow, Γ_i, is composed of m steps, and they are numbered in topological order in the range $[1..m]$. Each instance of this e2e flow is activated by a workload event e_{in} (periodic or sporadic) arriving with a minimum interval of T_i. Each event handler except the first is released when its predecessors have finished. The j-th step within the e2e flow Γ_i is denoted as τ_{ij}, and it has a worst-case execution time C_{ij} and a best-case execution time C_{ij}^b.

In the context of this thesis, three other event handlers are considered to represent the multipath case, which do not have runtime effects: *Fork*, *Join* and *Merge*. The $Fork$ event handler generates an event in each output whenever an input event is received. The $Join$ event handler generates an output event when all the associated input events have arrived, and similarly, the $Merge$ event handler generates an output when an input event arrives at any of its inputs. Therefore, a step may have more than one immediate predecessor and/or successor steps, i.e. the steps may receive more than one input event from different predecessor steps and, similarly, the output event of an step may be the input event for more than one step. The subset of steps immediately preceding the step τ_{ij} is named Γ_{ij}^{pred}, and similarly, Γ_{ij}^{succ} are those steps that are immediate successors of step τ_{ij}.

Within an e2e flow, each step may have a global deadline D_{ij}, which may be larger than the activation period, relative to the nominal activation time of the workload event (t_{in}, for the n-th instance). End-to-end deadlines are those timing requirements set on the final steps of e2e flows, and due to their multipath nature, there may be more than one timing requirement in the same flow. For each instance of a step τ_{ij}, the difference between its completion time and the nominal activation time of the workload event that triggered that instance of its e2e flow is called response time, and it can be obtained by schedulability analysis techniques. The worst-case response time is denoted as R_{ij}, and similarly the best-case response time is denoted as R_{ij}^b. Steps represent a utilization of the processing resource of $U_{ij} = C_{ij}/T_i$, which can also be expressed as a percentage if multiplied by 100. Following this notation, the utilization of an e2e flow U_{Γ_i} is the sum of the utilization of the steps of Γ_i, calculated as follows:

$$U_{\Gamma_i} = \sum_{\forall \tau_{ij} \in \Gamma_i} U_{ij} \tag{3.1}$$

In Figure 3.1, a workload event e_{in1}, which is represented by a down-pointing arrow, activates an step and then its output event forks, activating two steps, whose output events combine to activate a final step. Horizontal blue arrows represent precedence relations among event handlers.

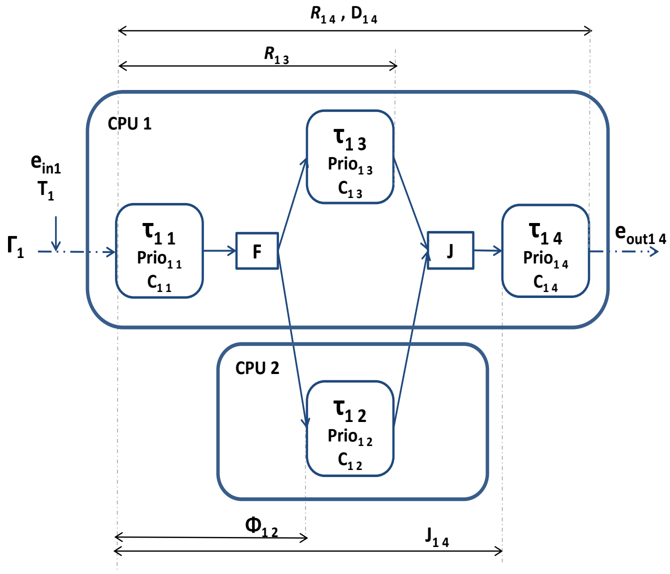


Fig. 3.1: Distributed multipath e2e flow

Workload events activating e2e flows and internal events activating handlers may have release jitter. The activation of internal events depends on the response times of the previous steps, which may vary for different instances. Therefore, any step τ_{ij} within an e2e flow may suffer release jitter up to a maximum of J_{ij}. Steps may also have an initial offset ϕ_{ij}, which is the minimum release time of the step τ_{ij}, relative to the nominal activation instant t_{in}. Hence, the release time for that step will be in the range of $[t_{in} + \phi_{ij}\ , t_{in} + max(\phi_{ij}, J_{ij})]$. For periodic e2e flows, deadlines, jitters and offsets can be larger than the periods of the e2e flow.

3.2 Physical architecture

This thesis considers a distributed architecture composed of processors, connected through one or more communications networks. Processors can be heterogeneous in terms of computation speed and memory resources. They provide hardware and software resources for task execution: sensors, actuators, memory, programs, libraries... They also host a real-time operating system that, among many other features, provides the capability of time-partitioning, where a fixed priority policy is used to schedule tasks within each partition. As they all refer to the same modeling element of a processing resource, in this thesis the terms "processor", "CPU" and "core" will be used indistinctly. The processor utilization U_{CPU_y} is the sum of the utilization of all the steps allocated to CPU_y:

$$U_{CPU_y} = \sum_{\forall \tau_{ij} \in CPU_y} U_{ij} \tag{3.2}$$

Regarding communications, we assume real-time networks in which worst-case message latencies can be measured or estimated. Normally, response times can be obtained in networks by using the same techniques used for processors. For instance, if the real-time communications network is an AFDX network [Aer09], the work in [GPH14] shows how to integrate the analysis of the messages in the network into the composite response time analysis technique described in Chapter 4. However, in the context of this work and with no loss of generality, it is assumed that networks are black boxes where each message is characterized by a minimum and a maximum latency. With this, the applicability of this model and its associated analysis and optimization tools to any distributed system whose network latencies are known, is guaranteed. The compositional approach [Riv+11] enables the integration of the network-level analysis with processor-level analysis.

3.3 Hierarchical scheduling

Processors make use of a real-time hierarchical scheduler, where a table-driven scheduling policy (time partitioning) is used for the primary scheduler. For the secondary scheduler, a preemptive FP policy is used, where $Prio_{ij}$ is the priority of step τ_{ij}, meaning the highest number, the highest priority. These priorities are valid in the context of each partition. Within a processor CPU_y, a temporal partition P_x is a set of n_x partition windows Win_{xk} within a *Major Frame* (MAF_y) that is cyclically repeated. Each partition window is defined by a start time S_{xk}, relative to the MAF, and a duration L_{xk}. Hence, partition windows contained in a temporal partition are defined as follows: $Win_{xk} = \{S_{xk}, L_{xk}\}$, k being in the range $1..n_x$. Figure 3.2 shows an example of a hierarchical scheduler where four tasks that belong to different e2e flows are allocated to P_1, and they are executed in P_1's windows according to their priorities.

The Available Utilization of the partition P_x, AU_{P_x}, is defined as the processing time allocated to P_x in its processor, which is in essence, following the terminology just presented, the sum of the utilization of all the temporal windows within the MAF, so:

$$AU_{P_x} = \sum_{\forall Win_{xk} \in P_x} L_{xk}/MAF_y \tag{3.3}$$

Following the description given for the utilization represented by each step, the Partition Utilization of P_x, U_{P_x} is defined as the sum of the utilization of all the steps contained in P_x:

$$U_{P_x} = \sum_{\forall \tau_{ij} \in P_x} U_{ij} \tag{3.4}$$

The overheads provoked by context switches at the primary scheduler are taken into account. This overhead is the time CS_y that CPU_y needs to load a partition context at the beginning of a partition window and to save it after execution finishes. In other words, it can be understood as a non-available CPU time whenever a partition window executes. For response time analysis purpose, this effect can be modeled by recording this unavailable time at the beginning of every partition window and subtracting this amount from the available CPU-time for that window, as shown in the example in Figure 3.3, where P_x executes on CPU_y within a MAF $MAF = 40\ ms$. In this example, a time partition is composed of two partition

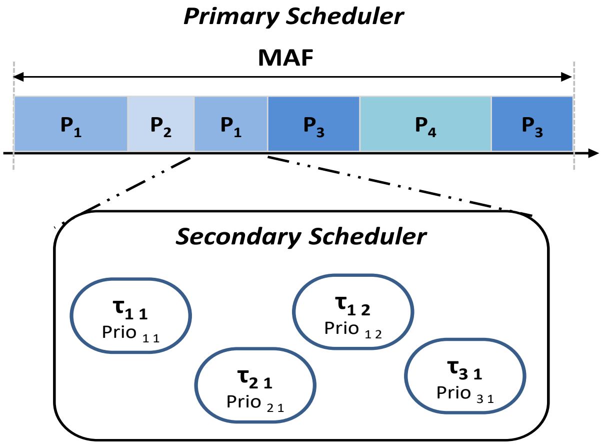


Fig. 3.2: Example of hierarchical scheduler

windows, and the effect of the context switch time of CS_y=1 ms at each window provokes that the total available time of the effective partition (P_x') is 2 ms less than the original P_x. Therefore, the effective partition window is defined as follows: $Win_{xk}' = \{\, S_{xk} + CS_y,\, L_{xk} - CS_y\}$.

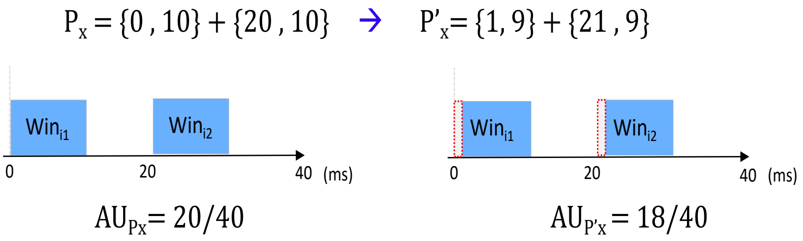


Fig. 3.3: Partition and effective partition with $CS_y = 1ms$

Table 3.1 contains a summary of the notation used for the real-time system model that describes the temporal behavior of an application. Although in this table all subscripts have been formalized, throughout this document some of them may be deliberately omitted if they are not strictly necessary, in order to ease readability.

3.4 Sensitivity analysis

Some of the algorithms proposed in this thesis, as well as some experimental results are based on parameters coming from performing a sensitivity analysis. Thus, similar to the sensitivity analysis proposed in MAST[1], a Slack Factor (SF) is defined as the factor by which the worst-case execution times of a step or a set of steps may be

[1] https://mast.unican.es/

increased while keeping the system schedulable (SF is then a positive value higher than 1), or decreased, in order to make the system schedulable (SF is then a positive value lower than 1). If SF is 1, the system is just schedulable. This definition of SF can be applied to different sets of steps, thus obtaining the following parameters that allow a comparison between different elements of the system or even the whole system:

- System Slack Factor (SSF), if all the steps in the systems are considered.
- Partition Slack Factor (PSF), if the SF is calculated by modifying only the steps of a particular partition.
- E2e Flow Slack Factor (FSF), if the SF is calculated by modifying only the steps of a particular e2e flow.
- Processor Slack Factor (CPUSF), if the SF is calculated by modifying only the steps of a particular processor.

These parameters enable the determination of how close a particular element is to schedulability, and they are obtained by iteratively applying a schedulability test. It is assumed that, when the SF calculation reaches a value below 0.001, it will be considered as zero and the calculation will stop, meaning that the system cannot be scheduled even if this element is removed.

Name	Notation	Description
e2e flow	Γ_i	End-to-end flow i, which represents the set of activities and actions executed in the system in response to a workload event
Workload event	e_{in}	Event that triggers the e2e flow number i
Period	T_i	Activation period or minimum interarrival time of the event that triggers the e2e flow i
Step	τ_{ij}	Step j in e2e flow i. It represents the execution of an operation in a processing resource, with some given scheduling parameters
Deadline	D_{ij}	Deadline of step j in e2e flow i. It is a timing requirement that must be satisfied, representing the maximum response time allowed for the step
Priority	$Prio_{ij}$	Priority of step j in e2e flow i. It is a scheduling parameter valid in the context of each partition. Highest value means highest priority.
Offset	Φ_{ij}	Offset of step j in e2e flow i. It is the minimum activation time of a step, relative to the nominal activation time of the workload event
Jitter	J_{ij}	Maximum release jitter that may be experienced in the activation of step j of e2e flow i. It is the maximum time that the activation of each step instance may be delayed.
Worst-case execution time	C_{ij}	Worst-case execution time of step j in e2e flow i
Best-case execution time	C^b_{ij}	Best-case execution time of step j in e2e flow i
Worst-case response time	R_{ij}	Worst-case response time of step j in e2e flow i. It is the maximum time elapsed from the nominal activation of the workload event and the completion of the step.
Best-case Response time	R^b_{ij}	Best-case response time of step j in e2e flow i. It is the minimum time elapsed from the nominal activation of the workload event and the completion of the step.
Processor	CPU_y	Processor y, providing HW/SW resources for task execution
Context switch overheads	CS_y	Time needed by processor y to handle the context switch of partitions
Major frame	MAF_y	Period of the cycle in processor y used by the primary scheduler, containing the temporal windows of all the partitions
Partition	P_x	Partition x, composed of a set of partition windows
Effective partition	P'_x	Effective Partition x, composed of a set of partition windows, after considering the context switch overheads
Partition window	Win_{xk}	Temporal window k in partition x
Effective partition window	Win'_{xk}	Temporal window k in partition x, after considering the context switch overheads
Start time	S_{xk}	Start time of the window k in partition x, within the MAF
Length	L_{xk}	Temporal length of the window k in partition x
Step utilization	U_{ij}	Usage of the processing element represented by the step τ_{ij}.
e2e flow utilization	U_{Γ_i}	The sum of the utilization of all the steps in an end-to-end flow i.
Processor utilization	U_{CPU_y}	The sum of the utilization of all the steps in CPU y.
Partition Utilization	U_{P_x}	The sum of the utilization of all the steps contained in P_x
Available Utilization	AU_{P_x}	Processing utilization time allocated to P_x

Tab. 3.1: Summary of notation

3.5 Modeling the industrial use-case

Considering the real-time model just defined in the previous section, the industrial use-case described in Chapter 1 can be modeled as shown in Figure 3.4. As can be seen, the complete subsystem is modeled as a single multipath e2e flow triggered by the workload event e_in1 representing the reception of a signal from the balise, which activates the signal-capturing task represented by step $\tau_{1\ 1}$ which, in turn, activates the three functionalities. Similarly to Figure 1.1 and for simplicity, only the real-time model of the EB functionality has been depicted in detail, even though the three functionalities presented in previous sections will be considered. In each processor two kinds of functions can be distinguished in response to the workload event: (1) those in charge of processing the information, voting and commanding the brakes for the EB functionality ($\tau_{1\ 2}$, $\tau_{1\ 12}$, $\tau_{1\ 13}$, $\tau_{1\ 6}$, $\tau_{1\ 10}$, $\tau_{1\ 11}$), and (2) the ones dealing with the communications to send and receive the messages ($\tau_{1\ 3}$, $\tau_{1\ 5}$, $\tau_{1\ 9}$, $\tau_{1\ 4}$, $\tau_{1\ 7}$, $\tau_{1\ 8}$). According to this classification, the steps will be allocated in two separate partitions for each processor: P_1 and P_3 hosting the steps for application processing and the I/O partition, P_2 and P_4, where the communication drivers are located hosting the communication steps. For the EB functionality, three messages are sent through the network, represented by steps $\tau_{1\ 38}$, $\tau_{1\ 39}$ and $\tau_{1\ 40}$. The other two functionalities follow the same scheme as the EB one: steps $\tau_{1\ 14}$ to $\tau_{1\ 25}$ model the RBC-CS functionality and steps $\tau_{1\ 26}$ to $\tau_{1\ 37}$ model the PV-DMI functionality. Steps $\tau_{1\ 41}$ to $\tau_{1\ 43}$ and $\tau_{1\ 44}$ to $\tau_{1\ 46}$ represent the messages transmitted in both functionalities.

The railway manufacturer provided average and worst-case execution times for each function measured in the current application, although these values cannot be shown for confidentiality reasons. For the experiemental evaluation of the techniques proposed in this thesis and based on these measures, worst-case execution times for each step which are of the same magnitude order as the real ones have been chosen. Best-case execution times have been assumed to be 50% of the worst-case ones. Furthermore, it is common that different functionalities have different workloads, and to highlight this effect we assume that the execution times of steps in RBC-CS and PV-DMI are five and ten times higher, respectively, than the ones in EB. All these values are shown in Table 3.2.

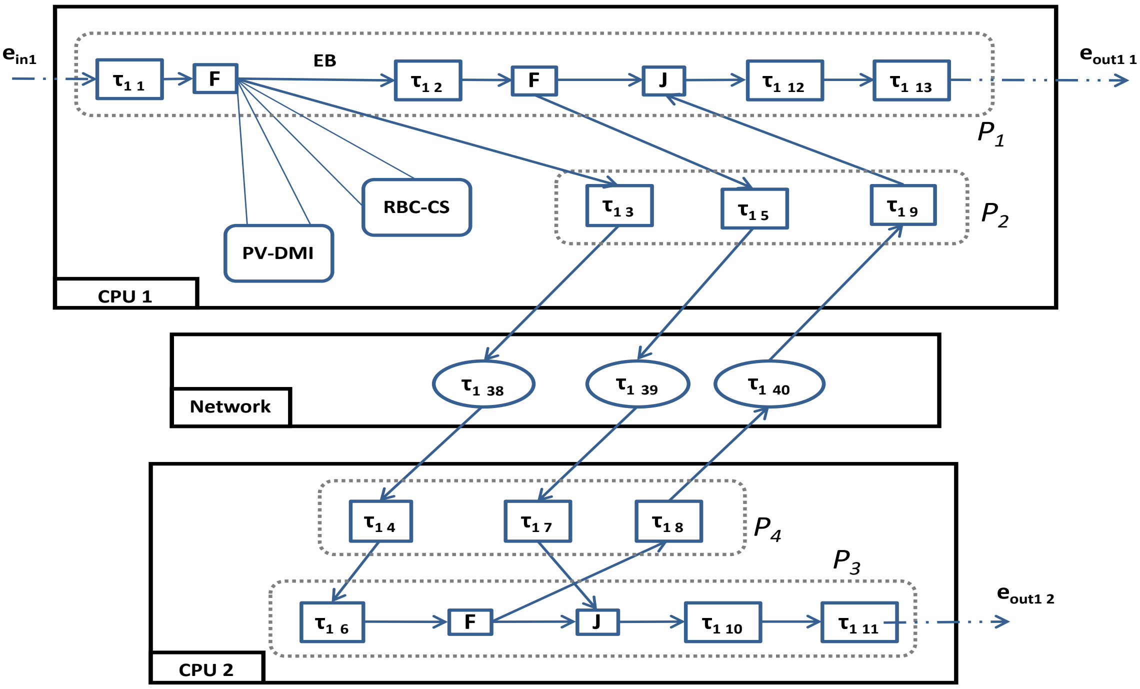


Fig. 3.4: Industrial use-case modeling (RBC-CS & PV-DMI not depicted for the sake of clarity)

Functionality	**Emergency-brake application - EB**												
τ_{ij}	$\tau_{1\ 1}$	$\tau_{1\ 2}$	$\tau_{1\ 3}$	$\tau_{1\ 4}$	$\tau_{1\ 5}$	$\tau_{1\ 6}$	$\tau_{1\ 7}$	$\tau_{1\ 8}$	$\tau_{1\ 9}$	$\tau_{1\ 10}$	$\tau_{1\ 11}$	$\tau_{1\ 12}$	$\tau_{1\ 13}$
C_{ij}	5	3	6	6	6	3	6	6	6	8	2	8	2
Functionality	**RBC Communication-session establishment - RBC-CS**												
τ_{ij}	-	$\tau_{1\ 14}$	$\tau_{1\ 15}$	$\tau_{1\ 16}$	$\tau_{1\ 17}$	$\tau_{1\ 18}$	$\tau_{1\ 19}$	$\tau_{1\ 20}$	$\tau_{1\ 21}$	$\tau_{1\ 22}$	$\tau_{1\ 23}$	$\tau_{1\ 24}$	$\tau_{1\ 25}$
C_{ij}	-	15	6	6	6	15	6	6	6	40	10	40	10
Functionality	**Parameter visualization - PV-DMI**												
τ_{ij}	-	$\tau_{1\ 26}$	$\tau_{1\ 27}$	$\tau_{1\ 28}$	$\tau_{1\ 29}$	$\tau_{1\ 30}$	$\tau_{1\ 31}$	$\tau_{1\ 32}$	$\tau_{1\ 33}$	$\tau_{1\ 34}$	$\tau_{1\ 35}$	$\tau_{1\ 36}$	$\tau_{1\ 37}$
C_{ij}	-	30	6	6	6	30	6	6	6	80	20	80	20

Tab. 3.2: Train signalling application (times in μs)

4 Response-time analysis

With the aim of assessing the schedulability of the optimization algorithms proposed in this thesis, a response-time analysis technique compliant with the proposed system model is necessary. Thus, in this chapter a new method for computing the worst-case response times of multipath e2e flows in hierarchically-scheduled and time-partitioned distributed real-time systems is presented. This technique is implemented in a prototype tool used for evaluating the industrial use-case and also general FP distributed real-time systems.

4.1 Response-time analysis of linear e2e flows

The offset-based analysis for time-partitioned systems [Pal+16] is based on the analysis for heterogeneous systems [Riv+11], where the response-time analysis is iteratively applied to each step τ_{ij} independently, using as input information an inherited offset ϕ'_{ij} and an inherited release jitter J'_{ij}. These inherited offsets and jitters are calculated for each step according to Equations 4.1 and 4.2 below, and they depend on the initial offset, ϕ_{ij}, and jitter, J_{ij}, and also on the worst- and best-case response times of the predecessor step in the e2e flow (τ_{ij-1}). As a result, the worst-case response times (R_{ij}) are obtained. A lower bound estimation of the best-case response time R^b_{ij} can be calculated by adding the best-case execution times of each step in the e2e flow up to τ_{ij}.

$$\phi'_{ij} = max(R^b_{ij-1}, \phi_{ij}) \tag{4.1}$$

$$J'_{ij} = J_{ij} + max(R_{ij-1}, \phi_{ij}) - \phi'_{ij} \tag{4.2}$$

Notice that these expressions are not applicable in multipath e2e flows, since there may be more than one task preceding the one under analysis.

4.2 Response-time analysis of multipath e2e flows

In the industrial use-case considered in this thesis, the model contains $Join$ and/or $Fork$ event handlers. These flow control handlers have no runtime effects (i.e., their execution time is zero) as they are just structural artifacts for handling events in MAST [Gon+01]. Therefore, in order to adapt the current compositional analysis to the multipath model, the inherited offsets and jitters obtained with Equations 4.1 and 4.2 at the input are directly propagated to all of its outputs in the case of a $Fork$ event handler. For this case, the analysis tools only need to be updated to perform this propagation to multiple outputs. However, for the *Join* event handlers, the previous equations are no longer valid as there is more than one predecessor step. Now, the correct procedure for propagating the values of the inherited offsets and jitters is shown.

Lemma1. For the analysis of a *Join* event handler in a multipath e2e flow, the inherited offset for step τ_{ij} that is the successor of the event handler is:

$$\phi'_{ij} = max(max_{\forall \tau_{ik} \in \Gamma_{ij}^{pred}} R_{ik}^{b}, \phi_{ij}) \tag{4.3}$$

where Γ_{ij}^{pred} is the set of predecessor steps of the $Join$ event handler.

Proof. The inherited offset ϕ'_{ij} is the minimum start time of τ_{ij}. By definition of the task model, this step cannot start before its initial offset ϕ_{ij} has elapsed since the arrival of the workload event. In addition, it cannot start before the *Join* event handler has generated its output. For this output to be generated it is necessary that all the *Join* input events have been generated. The minimum generation time for each of them is the best-case response time of their predecessor step. Therefore, by taking the maximum of all the R_{ik}^{b} for all k in the predecessors of the $Join$ handler the best-case generation time of its output event is obtained. Therefore, the lemma follows. □

Lemma 2. For the analysis of a $Join$ event handler in a multipath e2e flow, the inherited jitter for step τ_{ij} that is the successor of the event handler is:

$$J'_{ij} = J_{ij} + max(max_{\forall \tau_{ik} \in \Gamma_{ij}^{pred}} R_{ik}, \phi_{ij}) - \phi'_{ij} \tag{4.4}$$

Proof. The inherited jitter J'_{ij}, is the difference between the maximum and minimum start time of τ_{ij}. The minimum start time is given by the inherited offset ϕ'_{ij}. By definition of the task model, τ_{ij} cannot start before its initial offset ϕ_{ij} has elapsed since the arrival of the workload event. In addition, it cannot start before the $Join$ event handler has generated its output. For this output to be generated it is necessary that all the $Join$ input events have been generated. The maximum generation time for each of them is the worst-case response time of their predecessor steps. Therefore, by taking the maximum of all the R_{ik} for all k in the predecessors of the $Join$ handler we obtain the worst-case generation time of its output event. According to the task model, τ_{ij} may suffer an additional jitter bounded by its initial jitter J_{ij}. In the worst-case, there will be a delay equal to J_{ij} applied to the worst generation time of the $Join$ output event. Therefore, the lemma follows. □

Equations 4.3 and 4.4 are generalized expressions for the propagation of inherited offsets and jitters, which are the basis for applying offset-based analysis. Thus, existing analysis techniques can be directly applied to the model just by updating the propagation of offsets and jitters according to these equations. Furthermore, this solution can be applied not only to hierarchical time-partitioned systems but also to systems scheduled only by FP policy, since this would be a particular case where there is only one partition taking up the whole CPU.

4.2.1 Simple example

In order to explain how response-time analysis can be applied to the industrial use-case, a simple example is proposed. This enables the analysis of the key features, eg. fork-join multipath flows, considering a reduced number of steps and removing the communications network, without loss of generality. A simple example that can be solved manually provides the necessary intuition behind the theory.

Figure 4.1 shows a simple e2e flow consisting of four steps, two $Fork$ and two $Join$ event handlers. This e2e flow is distributed in two processors and executed in a single partition per processor. Both partitions have a 40 ms MAF and are composed of two partition windows: $Win_{11} = \{0, 10\}$ and $Win_{12} = \{20, 10\}$. The worst-case execution times in milliseconds and the priorities of the steps are also given in Figure 4.1. For simplicity, it is assumed that best-case and worst-case execution times are the same.

Table 4.1 shows the results of the analysis of the simple example using the time-partitioned analysis technique with the propagation of offsets and jitters updated according to Equations 4.3 and 4.4. The initial values for the analysis are shown, using the best-case results as the initial values for the worst-case response times. In addition, since the analysis technique is iterative, we show the results for the first

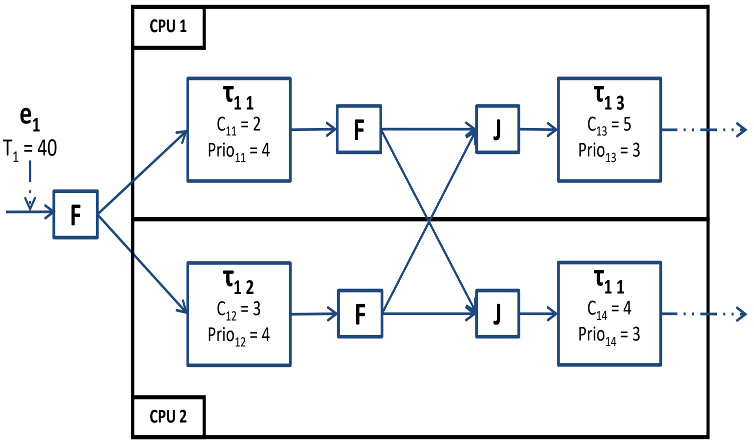


Fig. 4.1: Simple example

iteration. A second iteration gives the same results and thus no more iterations are required. Notice the effect of time-partitioning on the response times: due to those 10 ms windows in which the execution is interrupted in every cycle, the analysis must consider the worst-case scenario where the activation of all steps must be deferred until this unavailable window finishes and execution can be resumed. For more details on the effects of time partitioning refer to [Pal+16].

Step	**Initial**				**Iteration 1**			
	ϕ'	J'	R	R^b	ϕ'	J'	R	R^b
τ_{11}	0	0	2	2	0	0	**12**	**2**
τ_{12}	0	0	3	3	0	0	**13**	**3**
τ_{13}	3	0	8	8	3	10	**28**	**8**
τ_{14}	3	0	7	7	3	10	**27**	**7**

Tab. 4.1: Analysis results of the simple example (times in ms)

4.2.2 Implementation and tools

The analysis technique for time-partitioned systems [Pal+16] has been implemented in a prototype tool that includes the slanted offset-based technique [MN08]. The tool has been extended according to the results of this section to integrate the analysis of multipath e2e flows (with $Fork$ and $Join$ handlers), including the effects of the network as a black box. In addition, the multipath analysis has been implemented in the open-source tool MAST[1].

[1] https://mast.unican.es/

MAST version 1.5.1.0 supports multipath e2e flows with $Fork$, $Join$ and also $Merge$ control flow event handlers. The analysis of the $Merge$ event handler is directly supported by the proposed equations, since this event handler can be transformed into a linear model by replicating the sequence of steps following it a number of times equal to the number of predecessor steps [GPH00]. We have updated MAST by integrating Equations 4.3 and 4.4 in its offset analysis tools, in order to allow the application of these techniques to multipath e2e flows for FP scheduling. Until now, the analysis of these multipath e2e flows for the FP scheduling policy was only supported under the holistic technique, which is the most pessimistic for distributed systems. The offset-based techniques included in MAST that can now be applied to multipath flows are: the offset-based approximate analysis [PG98], the offset-based slanted analysis [MN08] and the offset-based brute force analysis [Tin94].

4.3 Industrial use-case evaluation

In this section, the proposed extensions to the response-time analysis of multipath end-to-end flows are evaluated. Since this work has been motivated by a real industrial need, the use-case described in Chapter 1 and modeled in Chapter 3 will be analyzed by applying the prototype tool.

As explained, the use-case functionalities have their deadlines imposed by the standard: for the three functionalities under analysis in this work, deadlines are all 1 s [UNI15]. According to the use-case model presented in Chapter 3, steps $\tau_{1\ 11}$, $\tau_{1\ 13}$, $\tau_{1\ 23}$, $\tau_{1\ 25}$, $\tau_{1\ 35}$ and $\tau_{1\ 37}$ in Figure 3.4 are the ones that must meet these deadlines.

The priority assignment for multipath e2e flows will be addressed in the next chapter, and for now, priorities are assigned to each step in the range [1..255] in order to evaluate the schedulability analysis proposed in this chapter. For the processing partitions (P_1 and P_3), the assignment is as follows: the EB functionality is considered to be the highest priority functionality, followed by the RBC-CS functionality and finally the PV-DMI. Within each functionality, priorities of the steps related to processing tasks are assigned following a decreasing order. In the I/O partitions (P_2 and P_4), there are only two tasks for all the functionalities: the receiving task with a higher priority than the sending task. This is a common way to implement communication drivers, and from the modeling point of view, this means that all the steps for sending are executed at the same priority (and similarly for receiving).

Functionality	Emergency-brake application - EB						
τ_{ij}	$\tau_{1\ 1}$	$\tau_{1\ 2}$	$\tau_{1\ 3}$	$\tau_{1\ 4}$	$\tau_{1\ 5}$	$\tau_{1\ 6}$	$\tau_{1\ 7}$
$Prio_{ij}$	220	219	100	200	100	215	200
R_{ij}	2455	4908	6184	9070	7394	11523	9070
τ_{ij}	-	$\tau_{1\ 8}$	$\tau_{1\ 9}$	$\tau_{1\ 10}$	$\tau_{1\ 11}$	$\tau_{1\ 12}$	$\tau_{1\ 13}$
$Prio_{ij}$	-	100	200	211	210	209	208
R_{ij}	-	12766	14409	13981	**16433**	16867	**19319**
Functionality	**RBC Communication-session establishment - RBC-CS**						
τ_{ij}	-	$\tau_{1\ 14}$	$\tau_{1\ 15}$	$\tau_{1\ 16}$	$\tau_{1\ 17}$	$\tau_{1\ 18}$	$\tau_{1\ 19}$
$Prio_{ij}$	-	119	100	200	100	115	200
R_{ij}	-	4933	6184	9070	7413	11548	9070
τ_{ij}	-	$\tau_{1\ 20}$	$\tau_{1\ 21}$	$\tau_{1\ 22}$	$\tau_{1\ 23}$	$\tau_{1\ 24}$	$\tau_{1\ 25}$
$Prio_{ij}$	-	100	200	111	110	109	8
R_{ij}	-	12785	14422	14040	**16500**	16914	**24374**
Functionality	**Parameter visualization - PV-DMI**						
τ_{ij}	-	$\tau_{1\ 26}$	$\tau_{1\ 27}$	$\tau_{1\ 28}$	$\tau_{1\ 29}$	$\tau_{1\ 30}$	$\tau_{1\ 31}$
$Prio_{ij}$	-	19	100	200	100	15	200
R_{ij}	-	7453	6184	9070	8702	16528	10333
τ_{ij}	-	$\tau_{1\ 32}$	$\tau_{1\ 33}$	$\tau_{1\ 34}$	$\tau_{1\ 35}$	$\tau_{1\ 36}$	$\tau_{1\ 37}$
$Prio_{ij}$	-	100	200	11	10	9	8
R_{ij}	-	17759	19390	21508	**23978**	24370	**26840**

Tab. 4.2: Response-time analysis of a train signalling application (times in μs)

Values of priorities of each step are shown in Table 4.2 together with their worst-case response times.

The application in use is based on a cyclic executive, and the intention is to explore the possibility of using time-partitioning with a twofold objective: (1) to maintain the application with the required SIL by guaranteeing that the time-requirements are met, and (2) to enable the use of the remaining capacity for other application components or even other applications. So, a tentative partition configuration that makes use of a small processing capacity, enough for the execution requirements, has been proposed. In Chapter 6 a partition scheduling algorithm, which will integrate a priority assignment stage, will be proposed.

For the evaluation of the proposed schedulability analysis technique, the following partition scheme is proposed. Partitions are allocated within a 10000 μs MAF. P_1 and P_3 are composed of four 50 μs windows arranged every 2500 μs, that is: $Win_{1\ 1} = \{0, 50\}$, $Win_{1\ 2} = \{2500, 50\}$, $Win_{1\ 3} = \{5000, 50\}$ and $Win_{1\ 4} = \{7500, 50\}$. P_2 and P_4 are defined by six windows of 25 μs executed every 1250 μs: $Win_{2\ 1} = \{1000, 25\}$, $Win_{2\ 2} = \{2250, 25\}$, $Win_{2\ 3} = \{3500, 25\}$, $Win_{2\ 4} = \{4750, 25\}$, $Win_{2\ 5} = \{6000, 25\}$, $Win_{2\ 6} = \{7250, 25\}$, $Win_{2\ 7} = \{8500, 25\}$ and $Win_{2\ 8} = \{9750, 25\}$. For now, context switch times are not considered. As mentioned in Chapter 3, this work does not consider a specific network, so it is modeled as a black box producing a bounded latency that can be measured in different ways, so this model is still realistic. In order to show the effects of the network on the analysis it is assumed that the maximum and minimum latencies for all messages are 400 μs and 40 μs, respectively, considering a low-loaded 100 Mb/s switched Ethernet network and 500Mb average message sizes.

Finally, Table 4.2 also shows the worst-case response times obtained by the analysis. As can be seen, deadlines are met comfortably, but the most promising fact is that these worst-case response times have been guaranteed with a partition configuration proposal that uses only 4% of the CPU. Other real-time applications allocated in other partitions could be analyzed separately, and in case any of them did not meet its deadlines, the results from the analysis could be used to reconfigure the partitioning scheme.

4.4 Response-time analysis performance

With the aim of completing the assessment on this new technique focusing now on general distributed systems scheduled by a FP policy, different logical sequences of steps will be analyzed by generating a synthetic application using the tool Task Graphs For Free (TGFF) [DRW98]. The DAG model employed in TGFF can be directly implemented in MAST and also in this prototype tool as all the necessary elements are available. Thanks to the contributions of this work, the results of worst-case response times of multipath flows between the holistic analysis [GPH00] and the offset-based slanted analysis [MN08] can be compared for the first time. In order to ensure that all experiments in this section are fully replicable by the real-time community, concrete instances of analysis problems are addressed, rather than providing massive results based on generic values.

An architecture composed of three multipath e2e flows and four processors has been designed. For the sake of simplicity, network connections, which can be modeled and analyzed along with processors, have not been included in this experiment. Each e2e flow has a different activation rate, and there may be more than one output and hence several deadline requirements. Step-to-processor mapping is performed randomly, the only restriction being that two consecutive steps are allocated to different processors. Steps' worst-case execution times are generated randomly in the range of [25,45] ms, and best-case execution times are assumed to be half of the worst-case ones. Priorities are assigned in a decreasing manner within each flow, and flows are prioritized with regard to their periods: the lower their period,

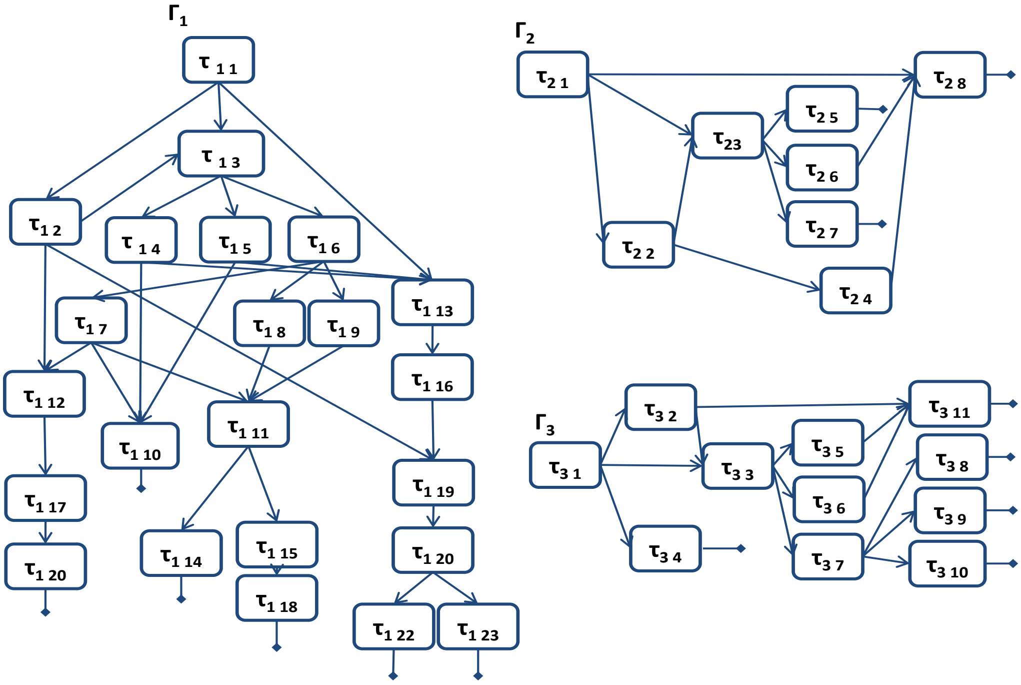


Fig. 4.2: Synthetic application generated with TGFF

the higher the priorities assigned to their steps. These decisions are taken in order to avoid assigning the same priorities to different steps, and bearing in mind that priority assignment and optimization is beyond the scope of this chapter. The complete configuration of the experiment is shown in Table 4. Results for both holistic and offset-based analysis are shown too: values corresponding to those steps with deadline requirements are highlighted in bold. Listing 4.1 shows the input parameters for the graph generation (readers are encouraged to visit the reference on this tool for a deeper understanding on DAG generation [DRW98]), and the generated flows have been depicted in Figure 4.2.

```
tg_cnt 3
task_cnt 15 2
task_degree 3 3
period_mul 0.5,0.7,1.5,1,0.3,5
tg_write
eps_write

table_cnt 1
type_attrib exec_time 35 10
trans_write
```

Listing 4.1: Input code for TGFF

Γ_1	**Period=1157.14**											
Step	$\tau_{1\ 1}$	$\tau_{1\ 2}$	$\tau_{1\ 3}$	$\tau_{1\ 4}$	$\tau_{1\ 5}$	$\tau_{1\ 6}$	$\tau_{1\ 7}$	$\tau_{1\ 8}$	$\tau_{1\ 9}$	$\tau_{1\ 10}$	$\tau_{1\ 11}$	$\tau_{1\ 12}$
CPU	1	2	3	4	1	2	3	4	1	2	2	4
C_{ij}	34.61	36.88	42.06	27.006	44.10	35.78	30.43	42.06	34.61	34.13	25.38	37.43
$Prio_{ij}$	23	22	21	20	19	18	17	16	15	14	13	12
R_{ij} (Hol)	246.41	496.03	733.67	888.8	1024.18	1019.07	1287.14	1259.99	1443.7	**1703.59**	1991.2	1691.96
R_{ij} (Off)	208.98	418.57	626.44	781.57	882.23	834.93	1031.17	1005.12	1043.91	**1350.71**	1507.12	1196.72
Step	$\tau_{1\ 13}$	$\tau_{1\ 14}$	$\tau_{1\ 15}$	$\tau_{1\ 16}$	$\tau_{1\ 17}$	$\tau_{1\ 18}$	$\tau_{1\ 19}$	$\tau_{1\ 20}$	$\tau_{1\ 21}$	$\tau_{1\ 22}$	$\tau_{1\ 23}$	-
CPU	1	3	3	4	1	2	3	4	1	2	3	-
C_{ij}	44.10	38.77	42.06	35.784	27.54	27.54	27.54	29.31	25.38	40.03	35.78	-
$Prio_{ij}$	11	10	9	8	7	6	5	4	3	2	1	-
R_{ij} (Hol)	1609.05	2464.51	2584.14	2087.08	2348.47	**3184.57**	**2918.61**	2974.18	3683.62	**4379.16**	**4661.58**	-
R_{ij} (Off)	1138.24	1813.96	2005.45	1452.37	1653.41	**2305.25**	**2045.7**	2053.73	2657.43	**3157.48**	**3286.54**	-
Γ_2	**Period=385.714**											
Step	$\tau_{2\ 1}$	$\tau_{2\ 2}$	$\tau_{2\ 3}$	$\tau_{2\ 4}$	$\tau_{2\ 5}$	$\tau_{2\ 6}$	$\tau_{2\ 7}$	$\tau_{2\ 8}$	-	-	-	-
CPU	1	2	3	3	1	2	4	4	-	-	-	-
C_{ij}	30.39	27.54	37.43	37.43	43.72	29.3	29.32	27.54	-	-	-	-
$Prio_{ij}$	42	41	40	39	38	37	36	35	-	-	-	-
R_{ij} (Hol)	30.39	57.94	95.37	132.8	**169.49**	152.23	**124.69**	**209.09**	-	-	-	-
R_{ij} (Off)	30.39	57.94	95.37	132.8	**139.09**	124.69	**124.69**	**160.34**	-	-	-	-
Γ_3	**Period=540**											
Step	$\tau_{3\ 1}$	$\tau_{3\ 2}$	$\tau_{3\ 3}$	$\tau_{3\ 4}$	$\tau_{3\ 5}$	$\tau_{3\ 6}$	$\tau_{3\ 7}$	$\tau_{3\ 8}$	$\tau_{3\ 9}$	$\tau_{3\ 10}$	$\tau_{3\ 11}$	-
CPU	1	2	4	3	1	2	3	4	1	2	3	-
C_{ij}	37.43	35.78	27.54	25.38	25.38	40.03	35.78	43.72	37.43	40.03	29.76	-
$Prio_{ij}$	34	33	32	31	30	29	28	27	26	25	24	-
R_{ij} (Hol)	111.55	204.19	288.59	**211.8**	425.53	421.26	424.62	**552.74**	**598.99**	**597.32**	**591.33**	-
R_{ij} (Off)	111.54	204.19	288.59	**211.8**	388.10	385.49	399.24	**499.82**	**510.79**	**496.13**	**492.73**	-

Tab. 4.3: Synthetic application analysis, times in ms. *(Hol) = Holistic Analysis, (Off) = Offset-based analysis*

It has been stated that the industrial use-case that has motivated this work represents a very low-loaded system. That is why now the evaluation of an application that makes use of a higher percentage of the available CPU time (an average of 51%) has been chosen, so that the experimental evaluation of this new analysis technique is complete.

As said before, the holistic analysis assumes that tasks are independent and the offset-based analysis reduces the pessimism of this approach [PG98], and that is exactly what can be observed in the results of this evaluation in Table 4.3: all the worst-case response time values obtained by this new technique are equal to or lower than those obtained by means of the holistic analysis. Moreover, it is common that in distributed systems deadlines are larger than the e2e flows' periods, so if in this case deadlines were three or even four times the periods, the offset-based analysis obtains schedulable solutions while the holistic analysis does not, as has been experimentally proved. The pessimism of the holistic analysis in comparison with the offset-based analysis has been represented in Figure 4.3, where the differences in percentage between the worst-case response times obtained with both analysis techniques have been plotted for each step (for the sake of clarity each flow has been plotted in a different colour). While high priority tasks may not be affected by the pessimism of

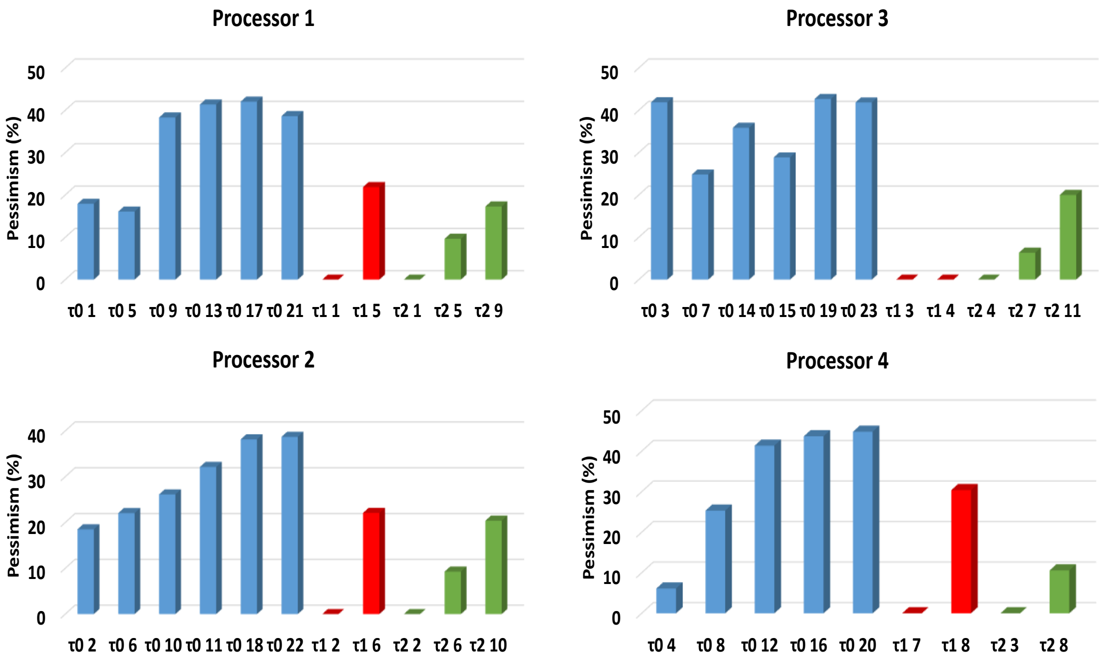


Fig. 4.3: Differences between worst-case response times obtained with the holistic and the offset-based analysis

the holistic analysis (a 0% of pessimism is obtained in the analysis of these tasks), low priority ones do clearly exhibit this effect as their worst-case response times are always higher than the ones obtained by this tool, showing up to 40% improvements in their worst-case response times. Therefore, the initial notion that the offset-based analysis applied to multipath flows would outperform the results from the holistic analysis, which was the only analysis technique available until now, is confirmed.

4.5 Conclusions

In this chapter a new schedulability analysis method for hierarchically-scheduled time-partitioned distributed real-time systems has been proposed, so that timing behavior of multipath end-to-end flows can be evaluated. This is motivated by a real industrial railway application and the need to accurately analyze its timing behavior, in order to perform a complete reconfiguration of its architecture and execution model. It is common practice that different manufacturers take part in the system design, integrating different applications in separate partitions. With this new analysis technique each of these complex applications can be analyzed separately and then integrated to check the overall schedulability. Moreover, thanks to this contribution on the extension of the offset-based analysis to support multipath flows, works such as [Ans+13] from the automotive domain are no longer obliged to relax task dependencies and simplify their models in order to make them linear.

5 Priority assignment

As a step forward in re-factoring railway signalling applications, the aim of this chapter is to find feasible priority assignment solutions for the kind of systems represented by the industrial use-case. Rather than implementing complex priority assignment algorithms that might entail long computation times, several state-of-the-art non-iterative algorithms will be adapted to the multipath model addressed in this work, motivated by the reasonably good behavior they exhibited in the literature for different scenarios [RG+16]. Their performance will be analyzed and compared by applying them (1) to the real industrial use-case, and (2) to a complex synthetic system that enables the exploration of their performance in a wide range of system configurations. The response-time analysis technique used to carry out the experimental evaluation is the offset-based technique proposed in Chapter 4.

5.1 Scheduling-parameter assignment overview

Scheduling-parameter assignment (priorities in FP schedulers or scheduling-deadlines in EDF schedulers) is vital in the design and development of real-time systems. Due to the large number of possible assignment combinations in distributed systems, this problem is considered NP-hard for non-trivial cases [TBW92], meaning that it may not be possible to find optimal solutions in polynomial time. That is why researchers have dedicated their efforts to developing algorithms that reach sub-optimal solutions in an acceptable computational time, and which are typically based on iterative optimization algorithms that improve their results at each iteration following some optimization criteria, which implies long computation times anyway.

In order to achieve the objectives mentioned in Section 1.3, a collection of non-iterative algorithms proposed in the literature for different application domains is selected. The same methodology as in [Riv+14] is followed, where the authors assign what they denominate Virtual Deadlines (VDs) to each step. These VDs are not timing requirements but just a mechanism to distribute the e2e deadline across all the steps of the e2e flow. The following algorithms have been selected:

- Ultimate Deadline (UD)

It is the simplest scheduling-parameter assignment algorithm, where the e2e deadline is assigned to all steps composing the e2e flow [Liu00]. It was used in [Riv+14] for VD assignment in linear e2e flows based on EDF schedulers.

$$VD_{ij} = D_i \tag{5.1}$$

where D_i refers to the e2e deadline of the linear e2e flow Γ_i.

- Effective Deadline (ED)

  The VD of a step according to the ED algorithm is the e2e deadline minus the sum of the worst-case execution times of its successor steps [Liu00] . In [Riv+14] it was also used for VD assignment, considering linear e2e flows scheduled by EDF policy.

$$VD_{ij} = D_i - \sum_{k=j+1}^{N_i} C_{ik} \tag{5.2}$$

where N_i is the index of the last step in a linear e2e flow.

- Proportional Deadline (PD)

  The e2e deadline is distributed/dealt among all the steps in the flow proportionally to their worst-case execution times and the sum of the worst-case execution times of all the steps of the flow [Liu00].

  In [Riv+14] the authors used this algorithm in non-synchronized linear distributed systems based on EDF, therefore when distributing the e2e deadline they assigned local scheduling deadlines to steps. These deadlines are referred to the event that activates that step, and are interpreted as local in the literature; it has been accepted that in linear e2e flows the sum of local deadlines should be the e2e deadline [SLB10]. On the contrary, if there is a global clock and all scheduling deadlines are referred to the workload event that activates the e2e flow, they are global deadlines. The authors showed that interpreting deadline distribution algorithms as local or global produces significant differences in response times.

$$VD_{ij} = \frac{C_{ij}}{\sum_{k=1}^{N_i} C_{ik}} * Di \tag{5.3}$$

- Normalized-Proportional Deadline (NPD)

  This algorithm is similar to PD, but it also considers the utilization of the processing element where it is hosted [Liu00]. It was also used in [Riv+14] for scheduling-deadline assignment in EDF systems, based on linear e2e flows:

$$VD_{ij} = Di * \frac{C_{ij} * UP_{ij}}{\sum_{k=1}^{N_i} C_{ik} * UP_{ik}} \tag{5.4}$$

  where UP_{ij} refers to the the utilization of the processor where τ_{ij} is hosted.

- Equal Slack (EQS)

  This algorithm was proposed for on-line deadline assignment in soft real-time distributed systems, based on EDF schedulers [KG93a]. Deadline assignment is performed by equally dividing the slack, defined as the difference between the deadline and the worst-case response time. An interpretation of this algorithm was performed by [RG+16] for off-line scheduling parameter assignment of distributed linear e2e flows. Since activation times are unknown for off-line schedulers, the authors assumed that such activations happened at time zero, and tasks' response times were assumed to be their worst-case execution times. Paradoxically, this algorithm, which is non-iterative, produced better results than iterative algorithms when deadlines were larger than activation periods [RG+16]. In that work, VDs are assigned as shown in Eq 5.5:

$$VD_{ij} = C_{ij} + \frac{D_i - \sum_{k=j}^{N_i} C_{ik}}{N_i - j + 1} \tag{5.5}$$

  There are three main elements in Eq. 5.5: 1) the worst-case execution time of the step under assignment, 2) the numerator term, where the sum of the worst-case execution times of all the steps from the step under analysis till the end of the e2e flow are subtracted from the e2e deadline, and 3) the denominator term, which is the relative position of the step counted from the end of the e2e flow.

- Equal Flexibility (EQF)

  This algorithm was also originaly proposed for on-line EDF scheduling [KG93a], and it is also based on dividing the slack, while the proportionality with respect to the execution times of the steps is maintained. To do so, the concept flexibility is defined as the ratio between the slack and the worst-case response time. This algorithm was also adapted for off-line scheduling of linear e2e flows in [RG+16], and similarly to the previous algorithm, it outperformed other iterative algorithms when deadlines are higher than activation periods and in those scenarios where the e2e flows had to transit through the same processor more than once.

$$VD_{ij} = C_{ij} + \left[D_i - \sum_{k=j}^{Ni} C_{ik} \right] * \left[\frac{C_{ij}}{\sum_{k=j}^{N_i} C_{ik}} \right] \tag{5.6}$$

Equation 5.6 is composed of three main elements: 1) the worst-case execution time of the step under assignment C_{ij}, 2) a factor where the execution times of all the successor steps from the step under assignment till the end of the e2e flow are subtracted from the e2e deadline, and 3) a proportionality factor between the worst-case execution time of the step under assignment and the sum of all the successor steps from the step under assignment till the end of the e2e flow.

Supported by these results from the background literature, the proposal is to adapt these algorithms to the system model addressed in this work, which includes multipath e2e flows and time-partitions.

5.2 Priority assignment in multipath e2e flows within time partitions

In order to find schedulable solutions to the problem addressed in this thesis, a two-step strategy will be followed. First, based on the algorithms described in Section 5.1, the proposed new algorithms will be formulated, in order to apply the previously reported state-of-the-art algorithms to multipath e2e flows within hierarchically scheduled time-partitioned architectures. These algorithms produce Virtual Deadlines (VDs). Then, VDs will be transformed into priorities in the second step of this proposal, following a Deadline Monotonic policy in the context of each partition.

5.2.1 Virtual Deadline assignment

To illustrate the Virtual Deadline assignment process, a simple yet paradigmatic example is depicted in Figure 5.1, where all of the challenging new features are

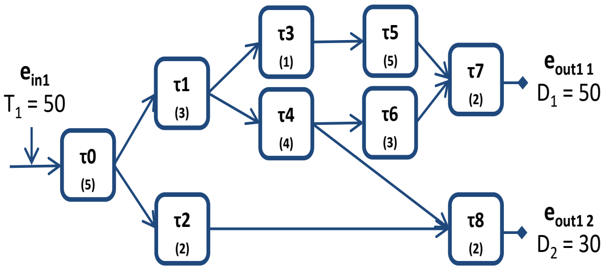


Fig. 5.1: Illustrative example

contained: a single workload event triggers the execution of a multipath e2e flow with different timing constraints at its output events. The number within brackets represents the worst-case execution time of each step. For the sake of clarity, a single processor without time partitioning is assumed for this illustrative example, although distributed architectures and time-partitions in them will be considered later. In the following lines the algorithms proposed for Virtual Deadline assignment are shown, and then Table 5.1 (located in section 5.2.2) details the results of their application. Each algorithm is presented with its pseudocode, showing how it has been implemented.

Ultimate Deadline (UD)

Due to the multipath nature of the industrial use-case, there may be more than one timing requirement at different outputs. Therefore, an adequate propagation of such deadlines must be done, guaranteeing that the effect of the most restrictive one is propagated through all the paths where it has an influence. This propagation can be seen in Algorithm 1.

Algorithm 1: Ultimate Deadline

```
Initialize all $VD_{ij}$ to inf.
for j ← $N_i$ to 1 in each $\Gamma_i$ do
  if $\nexists\Gamma_{ij}^{succ}$ then
    $VD_{ij} = D_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{succ}$ do
      if $VD_{ik} < VD_{ij}$ then
        $VD_{ij} = VD_{ik}$
      end if
    end for
  end if
end for
```

Effective Deadline (ED)

In addition to the issue of the different end-to-end deadlines, which applies here too, the execution time of the successor steps also influences the calculation of each VD. Therefore, for each step, the VD will be the most restrictive value of the difference between the successor´s VD and its worst-case execution time, as shown in Algorithm 2.

Algorithm 2: Effective Deadline

```
Initialize all $VD_{ij}$ to inf.
for j ← $N_i$ to 1 in each $\Gamma_i$ do
  if $\nexists \Gamma_{ij}^{succ}$ then
    $VD_{ij} = D_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{succ}$ do
      if $VD_{ik} - C_{ij} < VD_{ij}$ then
        $VD_{ij} = VD_{ik} - C_{ik}$
      end if
    end for
  end if
end for
```

Proportional Deadline (PD)

As previous experience in applying this algorithm with local and global deadlines corroborates that remarkable differences can be obtained in the schedulability of linear e2e flows [Riv+14], two versions of this algorithm for this system model are also proposed.

- Global version (PD_Global):

  Based on the algorithm proposed in [Liu00] and described in Section 5.1, the global version of the proportional deadline algorithm is proposed through Eq. 5.7, which retains the essence of the original algorithm:

$$VD_{ij} = Load_{ij} * F_{ij} \tag{5.7}$$

  where:

- $Load_{ij}$

  Is the accumulated load (sum of C_{ij}) from the workload event at each step. When there is more than one path, the highest possible value will be considered. This propagation is shown in step 1 of Algorithm 3.

- F_{ij}

  Represents the proportionality factor between the e2e deadline and the accumulated load at each step. To determine this value, the e2e deadline D_{ij} is divided by the term $Load_{ij}$ at all output steps with an e2e deadline. Then, it is propagated backwards, and in those steps with more than one predecessor step (where different e2e deadlines may have effect) the highest value of F_{ij} is propagated (so that the most restrictive VD is produced). This is shown in step 2 of Algorithm 3.

- Local version (PD_Local)

  This algorithm turns global deadlines obtained by Algorithm 3 into local deadlines. To do so, we invert the notion of local deadlines explained before, where the summation of local deadlines provided the e2e deadline in linear e2e flows: from each of the output steps in the e2e flow, the local VD of each step is obtained by subtracting the value of the global VD from the predecessor step. If there is more than one, the most restrictive, i.e. lowest value, VD is assigned. The algorithm that converts global VDs into local ones is described in Algorithm 4.

Normalized Proportional Deadline (NPD)

This algorithm is similar to the PD algorithm, but in this case a normalization factor is applied that considers the utilization of the resource where steps are hosted. Two versions of the algorithm are proposed too, considering Global and Local VDs:

- Global Version (NPD_Global)

  Following the same criterion applied to the PD algorithms, Eq.5.8, which is based on the original formulation, is proposed, and then the calculation of each factor that composes it is explained in Algorithm 5.

$$VD_{ij} = Load_{ij}^{'} * F_{ij}^{'} \tag{5.8}$$

Algorithm 3: Proportional Deadline (PD_Global)

```
Step 1:
Initialize all $Load_{ij} = 0$
for j $\leftarrow$ 1 to $N_i$ in each $\Gamma_i$ do
  if $\nexists\Gamma_{ij}^{pred}$ then
    $Load_{ij} = C_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ do
      if $Load_{ik} + C_{ij} > Load_{ij}$ then
        $Load_{ij} = Load_{ik} + C_{ij}$
      end if
    end for
  end if
end for

Step 2:
Initialize all $F_{ij}$ to inf.
for j $\leftarrow N_i$ to 1 in each $\Gamma_i$ do
  if $\nexists\Gamma_{ij}^{succ}$ then
    $F_{ij} = D_{ij}/Load_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{succ}$ do
      if $F_{ik} < F_{ij}$ then
        $F_{ij} = F_{ik}$
      end if
    end for
  end if
end for

Step3:
for each $\tau_{ij}$ in each $\Gamma_i$ do
  Calculate VDs by Eq. 5.7
end for
```

Algorithm 4: Turn Global VD into Local VD

```
for j $\leftarrow N_i$ to 1 in each $\Gamma_i$ do
  $VD_{min} = VD_{ij}$
  for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ do
    if $VD_{ij} - VD_{ik} < VD_{ij}$ then
      $VD_{min} = VD_{ij} - VD_{ik}$
    end if
  end for
  $VD_{ij} = VD_{min}$
end for
```

where

  - $Load'_{ij}$

    Is the accumulated load from the workload event at each step. Since we are addressing partitioned systems, this factor will refer to the utilization of the partition where the step is allocated (P_x). Thus, the accumulated value is $C_{ij} * U_{P_x}$. When there is more than one predecessor step, the maximum value is propagated, as shown in the step 1 of Algorithm 5.

  - F'_{ij}

    Is the proportionality factor between the e2e deadline and the accumulated load at each step. It is calculated in the same way as in the PD_Global algorithm, as described in step 2 of Algorithm 5.

- Local Version (NPD_Local)

  In order to develop the local version of the Normalized Proportional Deadline algorithm, Algorithm 4 will be applied, which turns Global VDs into Local ones.

Equal Slack (EQS)

When Eq. 5.5 was formulated in [RG+16], the authors considered linear systems where there is only a single e2e deadline. However in the model considered in this work, the relative position of the step within the e2e flow becomes non-trivial due to the multipath structures. Therefore, an equivalent equation (Eq. 5.9), which is similar in its structure, is proposed, and in Algorithm 6 the calculation of the terms that compose the equation is detailed.

$$VD_{ij} = C_{ij} + \frac{H1_{ij}}{H2_{ij}} \tag{5.9}$$

where $H1_{ij}$ refers to the numerator term from Eq. 5.5 and $H2_{ij}$ reflects the denominator term. When calculating these terms, if there is more than one predecessor step we will consider the one that produces the highest $H1_{ij}/H2_{ij}$ value, with the aim of obtaining the most restrictive VD.

Algorithm 5: Normalized Proportional Deadline (NPD_Global)

```
**Step 1:**
Initialize all $Load'_{ij} = 0$
for j $\leftarrow$ 1 to $N_i$ in each $\Gamma_i$ do
  for each $P_x$ do
    if $\tau_{ij} \in P_x$ then
      if $\nexists\Gamma_{ij}^{pred}$ then
        $Load'_{ij} = C_{ij} * U_{P_x}$
      else
        for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ do
          if $Load'_{ik} + C_{ij} * U_{P_x} > Load'_{ij}$ then
            $Load'_{ij} = Load'_{ik} + C_{ij} * U_{P_x}$
          end if
        end for
      end if
    end if
  end for
end for

Step 2:
Initialize all $F'_{ij}$ to inf.
for j $\leftarrow$ $N_i$ to 1 in each $\Gamma_i$ do
  if $\nexists\Gamma_{ij}^{succ}$ then
    $F'_{ij} = D_{ij}/Load'_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{succ}$ do
      if $F'_{ik} < F'_{ij}$ then
        $F'_{ij} = F'_{ik}$
      end if
    end for
  end if
end for

Step3:
for each $\tau_{ij}$ in each $\Gamma_i$ do
  Calculate VDs by Eq. 5.8
end for
```

Algorithm 6: Equal Slack

```
Initialize all $H1$ to inf. and $H2$ to 0.0
for j $\leftarrow N_i$ to 1 in each $\Gamma_i$ do
  if $\nexists\Gamma_{ij}^{succ}$ then
    $H1_{ij} = D_{ij} - C_{ij}$
    $H2_{ij} = 1$
    Calculate $VD_{ij}$ through Eq. 5.9

  else

    $H1_{ij} = H1_{ij} - C_{ij}$
    $H2_{ij} = H2_{ij} + 1$

    Calculate $VD_{ij}$ through Eq. 5.9

  end if
  for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ do
    if $H1_{ik}/H2_{ik} > H1_{ij}/H2_{ij}$ then
      $H1_{ik} = H1_{ij}$
      $H2_{ik} = H2_{ij}$
    end if
  end for
end for
```

Equal Flexibility (EQF)

The formulation in Eq. 5.6 also considered linear e2e flows, and the sum of worst-case execution times from the step under assignment becomes non-trivial too, as there is more than a single path to take into account. Therefore, Eq. 5.10 that retains the structure of Eq. 5.6, is proposed, and in Algorithm 7, details about this interpretation are given. The $Q1_{ij} * Q2_{ij}$ that produces the lowest VD in those cases where there is more than one predecessor step will also be chosen.

$$VD_{ij} = C_{ij} + Q1_{ij} * Q2_{ij} \tag{5.10}$$

Algorithm 7: Equal Flexibility

```
Initialize all $Q_1$ and $Q_2$ to inf.
for j $\leftarrow N_i$ to 1 in each $\Gamma_i$ do
  if $\nexists\Gamma_{ij}^{succ}$ then
    $Q1_{ij} = D_{ij} - C_{ij}$
    $Q2_{ij} = 1$

    Calculate $VD_{ij}$ through Eq. 5.10

  else
    $Q1_{ij} = Q1_{ij} - C_{ij}$
    $Q2_{ij} = \frac{C_{ij}}{Q2_{ij}} + C_{ij}$

    Calculate $VD_{ij}$ through Eq. 5.10

  end if
  for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ do
    if $Q1_{ik} * Q2_{ik} > Q1_{ij} * Q2_{ij}$ then
      $Q1_{ik} = Q1_{ij}$
      $Q2_{ik} = Q2_{ij}$
    end if
  end for
end for
```

5.2.2 Virtual Deadline transformation into priorities

As said before, the second stage of this priority assignment strategy is to transform the Virtual Deadlines into priorities. To do so, priorities in the context of each

Simple Example		$\tau_{1\,1}$	$\tau_{1\,2}$	$\tau_{1\,3}$	$\tau_{1\,4}$	$\tau_{1\,5}$	$\tau_{1\,6}$	$\tau_{1\,7}$	$\tau_{1\,8}$	$\tau_{1\,9}$
UD	VD$_{ij}$	30	30	30	50	30	50	50	50	30
	Prio$_{ij}$	9	8	7	4	6	3	2	1	5
	R$_{ij}$	5	8	10	17	14	22	25	**27**	**16**
ED	VD$_{ij}$	21	24	28	43	28	48	48	50	30
	Prio$_{ij}$	9	8	7	4	6	3	2	1	5
	R$_{ij}$	5	8	10	17	14	22	25	**27**	**16**
PD_Global	VD$_{ij}$	10.71	17.14	15	26.47	25.71	41.17	44.11	50	30
	Prio$_{ij}$	9	7	8	5	6	3	2	1	4
	R$_{ij}$	5	10	7	15	14	22	25	**27**	**17**
PD_Local	VD$_{ij}$	10.71	6.42	4.28	9.32	8.57	14.7	18.4	5.88	4.28
	Prio$_{ij}$	3	6	9	4	5	2	1	7	8
	R$_{ij}$	19	28	21	37	36	45	48	**50**	**38**
NPD_Global	VD$_{ij}$	10.71	17.14	15	26.47	25.71	41.17	44.11	50	30
	Prio$_{ij}$	9	7	8	5	6	3	2	1	4
	R$_{ij}$	5	10	7	15	14	22	25	**27**	**17**
NPD_Local	VD$_{ij}$	10.71	6.42	4.28	9.32	8.57	14.7	18.4	5.88	4.28
	Prio$_{ij}$	3	6	9	4	5	2	1	7	8
	R$_{ij}$	19	28	21	37	36	45	48	**50**	**38**
EQS	VD$_{ij}$	11.6	12.5	15	15	17.66	26.5	25.5	50	30
	Prio$_{ij}$	9	8	7	6	5	3	4	1	2
	R$_{ij}$	5	8	10	11	15	23	18	**27**	**25**
EQF	VD$_{ij}$	18.81	19.57	19.33	23.9	23.2	40.83	36.75	50	30
	Prio$_{ij}$	9	7	8	5	6	2	3	1	4
	R$_{ij}$	5	10	7	15	14	25	20	**27**	**17**

Tab. 5.1: Virtual Deadlines, priorities and worst-case response times for each algorithm, applied to the multipath e2e flow depicted in Figure 5.1

partition are assigned following a deadline monotonic order, which assigns the highest priority to the step with the smallest Virtual Deadline.

Being a multipath architecture, it is likely that the same Virtual Deadline is assigned to more than one step, mostly depending on the worst-case execution times of steps and also the e2e deadlines. In fact, there are cases where this kind of tie happens in a generalized manner, for instance: in the ED algorithm, the same VD will be assigned to all the steps preceding a Join event handler. If these steps are hosted in the same partition, assigning the same VD implies assigning the same priority, which is undesirable in the response time analysis technique applied in this work. In the absence of a clear criterion to solve such ties, the following method is proposed: steps are processed following their index order, and they are sorted in a non-decreasing order according to their VDs. Then, priorities are assigned following this order decreasingly. Thus, the same priority is never assigned to more than one step in the same partition. An optimized strategy for solving ties in priority assignment is a subject for future work.

Considering the illustrative example in Figure 5.1, Table 5.1 shows the different priority assignments obtained by applying the proposed algorithms. The response times obtained by applying the analysis technique are also shown for all steps. There are several remarkable conclusions to be mentioned. First, due to the nature of the UD algorithm, many VDs in the e2e flow are the same, which would lead to equal priorities if we did not solve these ties in some way. ED, PD_Local, NPD_Local and EQS also deal with ties, and they are solved as explained before. Second, according to the worst-case response times of steps $\tau_{1\ 8}$ and $\tau_{1\ 9}$, the best priority assignment algorithms are UD and ED, although PD_Global and EQF algorithms also obtain good results. However, as will be shown later, this should not be taken as representative for all situations, as there are other algorithms that exhibit better behavior in other different system configurations. Finally, it should be noticed that the VDs and hence the priority assignments given by PD_Global and PD_Local are the same as the ones produced by NPD_Global and NPD_Local respectively. This is an expected result for this simple example since all the steps are hosted in the same partition, and therefore the normalization factor has no effect. Later experiments will show that none of those algorithms produce the same VDs when e2e flows traverse more than one partition.

5.3 Evaluation of the priority assignment algorithms

In this section the proposed algorithms are evaluated in different scenarios. First, they are applied to the industrial use-case that motivates this thesis. Then, a more general evaluation is performed by generating synthetic e2e flows with a wide range of activation patterns and deadline requirements.

5.3.1 Industrial use-case

The resulting priority assignments are shown in Table 5.2. For all cases, the highest number means the highest priority, and as explained before, priorities are valid in the context of each partition. As in the previous example, here the different assignments obtained by the algorithms can be seen. The schedulability analysis introduced in Section 4 is applied for each priority assignment, and the worst-case response times obtained for the steps with an e2e deadline are compiled in Table 5.3. These steps are $\tau_{1\ 11}$ and $\tau_{1\ 13}$ for the emergency brake functionality, $\tau_{1\ 23}$ and $\tau_{1\ 25}$ for the RBC Communication-session establishment functionality and $\tau_{1\ 35}$ and $\tau_{1\ 37}$ for the parameter visualization functionality.

Functionality	**Emergency Brake - EB**												
τ_{ij}	$\tau_{1\ 1}$	$\tau_{1\ 2}$	$\tau_{1\ 3}$	$\tau_{1\ 4}$	$\tau_{1\ 5}$	$\tau_{1\ 6}$	$\tau_{1\ 7}$	$\tau_{1\ 8}$	$\tau_{1\ 9}$	$\tau_{1\ 10}$	$\tau_{1\ 11}$	$\tau_{1\ 12}$	$\tau_{1\ 13}$
C_{ij}	5	3	6	6	6	3	6	6	6	8	2	8	2
Priority Assignment													
UD	10	9	9	9	8	9	8	7	7	8	7	8	7
ED	10	7	3	3	2	7	1	2	1	4	3	4	3
PD_Global	10	9	7	3	3	7	1	2	1	4	3	4	3
PD_Local	10	6	2	2	1	8	1	3	3	3	9	3	9
NPD_Global	10	9	9	9	6	9	6	5	5	8	3	7	3
NPD_Local	10	4	1	3	2	8	1	2	3	3	9	3	9
EQS	10	9	7	7	4	9	1	4	1	6	3	6	3
EQF	9	6	4	6	9	6	1	9	1	9	3	10	3

Functionality	**RBC Communication-session establishment - RBC-CS**											
τ_{ij}	$\tau_{1\ 14}$	$\tau_{1\ 15}$	$\tau_{1\ 16}$	$\tau_{1\ 17}$	$\tau_{1\ 18}$	$\tau_{1\ 19}$	$\tau_{1\ 20}$	$\tau_{1\ 21}$	$\tau_{1\ 22}$	$\tau_{1\ 23}$	$\tau_{1\ 24}$	$\tau_{1\ 25}$
C_{ij}	15	6	6	6	15	6	6	6	40	10	40	10
Priority Assignment												
UD	6	6	6	5	6	5	4	4	5	4	5	4
ED	8	6	6	5	8	4	5	4	5	2	5	2
PD_Global	8	8	8	5	8	5	4	2	5	2	5	2
PD_Local	5	5	5	4	5	4	6	6	2	7	2	8
NPD_Global	8	8	8	4	7	4	3	3	5	2	5	2
NPD_Local	5	4	5	5	5	4	6	6	2	7	2	8
EQS	8	8	8	5	8	2	5	2	5	2	5	2
EQF	5	3	7	8	5	2	5	5	8	2	8	2

Functionality	**Parameter visualization - PV-DMI**											
τ_{ij}	$\tau_{1\ 26}$	$\tau_{1\ 27}$	$\tau_{1\ 28}$	$\tau_{1\ 29}$	$\tau_{1\ 30}$	$\tau_{1\ 31}$	$\tau_{1\ 32}$	$\tau_{1\ 33}$	$\tau_{1\ 34}$	$\tau_{1\ 35}$	$\tau_{1\ 36}$	$\tau_{1\ 37}$
C_{ij}	30	6	6	6	30	6	6	6	80	20	80	20
Priority Assignment												
UD	3	3	3	2	3	2	1	1	2	1	2	1
ED	9	9	9	8	9	7	8	7	6	1	6	1
PD_Global	7	9	9	6	9	7	6	4	6	1	6	1
PD_Local	4	8	8	7	4	7	9	9	1	6	1	7
NPD_Global	6	7	7	2	6	2	1	1	4	1	4	1
NPD_Local	6	9	9	7	4	7	8	8	1	6	1	7
EQS	7	9	9	6	7	3	6	3	4	1	4	1
EQF	4	2	8	7	4	3	4	6	7	1	7	1

Tab. 5.2: Priority assignment for the train signalling application (C_{ij} in μs)

Railway Signalling App	EB		RBC-CS		PV-DMI	
	$\tau_{1\ 11}$	$\tau_{1\ 13}$	$\tau_{1\ 23}$	$\tau_{1\ 25}$	$\tau_{1\ 35}$	$\tau_{1\ 37}$
Worst-case Response Times (ms)						
UD	12.28	14.74	12.37	14.83	19.85	22.31
ED	17.34	19.78	17.34	19.79	17.37	19.82
PD_Global	17.37	19.78	17.34	19.79	17.37	19.82
PD_Local	14.84	17.31	16.12	18.59	21.11	23.58
NPD_Global	14.89	17.33	14.9	17.34	17.37	19.81
NPD_Local	16.09	18.52	17.35	18.6	22.32	23.59
EQS	14.91	17.34	14.92	17.35	17.39	19.82
EQF	21.16	23.63	21.17	23.64	23.65	26.11

Tab. 5.3: Worst-case response times of the railway signalling application

	SSF
UD	81.87
ED	88.7
PD_Global	88.33
PD_Local	66.25
NPD_Global	87.5
NPD_Local	68.75
EQS	86.25
EQF	51.14

Tab. 5.4: SSF for each algorithm applied in the industrial use-case

In Table 5.3, the lowest response time for the whole application is obtained by the NPD_Global algorithm, as it completes its execution in 19.81 ms in the worst-case scenario, and ED, PD_Global and EQS show a very similar performance (19.82 ms in the worst-case scenario). Taking a look at each functionality independently, it is remarkable that with UD, PD_Local and NPD_Local algorithms the EB functionality finishes its execution several ms before the others. This, however, is achieved by penalizing the execution of the other two functionalities, so in those cases where applications must meet different deadline requirements, the collection of algorithms proposed in this work can provide different alternatives for their design.

Focusing only on the response times may not provide an adequate view of the system schedulability if there are several outputs in response to the same input event, which is actually the case described in this application. It is convenient to perform a sensitivity analysis through the calculation of the System Slack Factor (SSF) as has been defined in Section 3.4.

In this use-case, this factor can give designers an insight into how much the system could grow (in terms of utilization) when applying the different priority assignment algorithms. The SSFs calculated for each algorithm are shown in Table 5.4: even if NPD_Global seemed to be the best algorithm for the industrial use-case in terms of response times, the priority assignment that lets the system load grow the most is ED, closely followed by PD_Global and NPD_Global. The fact that the SSFs are so high can be explained by highlighting that the system represents a very low load and therefore execution times can be greatly increased.

5.3.2 Performance evaluation

After applying all the proposed algorithms to the industrial example, their behavior has to be assessed when external conditions differ from the ones that characterize this use-case, in order to get a deeper view of their behavior. Instead of generating a huge number of synthetic task sets to perform the evaluation, all the experiments should be reproducible by the research community and therefore a small-sized task set will be generated, which includes most of the representative features relevant for the system model this work is targeting. They will be generated with the tool TGFF [DRW98], which can be directly transformed to the system model described in this thesis and then processed by the developed tools. Ten different step-sequences are generated and depicted in Figure 5.2. Random worst-case execution times will be generated in a [0,20] ms range (note that a 1/3 scaling factor is later applied in order to produce feasible utilizations). These e2e flows will allow the following features to be tested:

- Activation periods: a wide range of activation periods will be tested, from 50 ms to 1 s. These values are within the ranges of the sampling frequency of automotive subsystems [SD07] and the typical minimum inter-arrival times of balises in the railway domain.

- Deadlines: Different deadline requirements (relative to the periods) will be evaluated. The most restrictive requirements are normally found in certification standards, although we will consider more relaxed deadlines as distributed systems normally have deadlines larger than periods.

- Number of processors: the behavior of the algorithms will be analyzed first by assigning 80% of available utilization to a single partition allocated to a single processor, and then by distributing this available utilization into 2, 4, 8 and 10 processors (with one partition each) having the same total available utilization (this means that when testing 2 processors, partitions within them will get 40% of available utilization, and so on). Step-to-core mapping is performed randomly, but two consecutive steps will not be allowed to be assigned to the same processors, if possible. The utilization of all partitions shall be kept in a fair balance so that the pessimism that unbalanced loads produce is minimized.

Listing 5.1 shows the input code for generating synthetic e2e flows with TGFF, and finally step-to-processor mapping is shown for each number of processors in Table 5.5.

Number of Processors	Step-to-Processor Mapping									
1	$\tau_{1\,1}$ $\tau_{1\,2}$ $\tau_{1\,3}$ $\tau_{1\,4}$ $\tau_{1\,5}$ $\tau_{1\,6}$ $\tau_{1\,7}$ $\tau_{2\,1}$ $\tau_{2\,2}$ $\tau_{2\,3}$ $\tau_{2\,4}$ $\tau_{2\,5}$ $\tau_{3\,1}$ $\tau_{3\,2}$ $\tau_{3\,3}$ $\tau_{3\,4}$ $\tau_{3\,5}$ $\tau_{3\,6}$ $\tau_{3\,7}$ $\tau_{3\,8}$ $\tau_{4\,1}$ $\tau_{4\,2}$ $\tau_{4\,3}$ $\tau_{4\,4}$ $\tau_{4\,5}$ $\tau_{5\,1}$ $\tau_{5\,2}$ $\tau_{5\,3}$ $\tau_{5\,4}$ $\tau_{5\,5}$ $\tau_{5\,6}$ $\tau_{5\,7}$ $\tau_{5\,8}$ $\tau_{5\,9}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{6\,2}$ $\tau_{6\,3}$ $\tau_{6\,4}$ $\tau_{6\,5}$ $\tau_{7\,1}$ $\tau_{7\,2}$ $\tau_{7\,3}$ $\tau_{7\,4}$ $\tau_{7\,5}$ $\tau_{7\,6}$ $\tau_{7\,7}$ $\tau_{7\,8}$ $\tau_{8\,1}$ $\tau_{8\,2}$ $\tau_{8\,3}$ $\tau_{8\,4}$ $\tau_{8\,5}$ $\tau_{8\,6}$ $\tau_{8\,7}$ $\tau_{9\,1}$ $\tau_{9\,2}$ $\tau_{9\,3}$ $\tau_{9\,4}$ $\tau_{9\,5}$ $\tau_{9\,6}$ $\tau_{9\,7}$ $\tau_{9\,8}$ $\tau_{10\,1}$ $\tau_{10\,2}$ $\tau_{10\,3}$ $\tau_{10\,4}$ $\tau_{10\,5}$ $\tau_{10\,6}$ $\tau_{10\,7}$									
2	$\tau_{1\,1}$ $\tau_{1\,3}$ $\tau_{1\,5}$ $\tau_{1\,7}$ $\tau_{2\,2}$ $\tau_{2\,4}$ $\tau_{3\,1}$ $\tau_{3\,3}$ $\tau_{3\,5}$ $\tau_{3\,7}$ $\tau_{4\,2}$ $\tau_{4\,4}$ $\tau_{5\,1}$ $\tau_{5\,3}$ $\tau_{5\,5}$ $\tau_{5\,7}$ $\tau_{5\,9}$ $\tau_{6\,2}$ $\tau_{6\,4}$ $\tau_{7\,1}$ $\tau_{7\,3}$ $\tau_{7\,5}$ $\tau_{7\,7}$ $\tau_{8\,2}$ $\tau_{8\,4}$ $\tau_{8\,6}$ $\tau_{9\,1}$ $\tau_{9\,3}$ $\tau_{9\,5}$ $\tau_{9\,7}$ $\tau_{10\,2}$ $\tau_{10\,4}$ $\tau_{10\,6}$	$\tau_{1\,2}$ $\tau_{1\,4}$ $\tau_{1\,6}$ $\tau_{2\,1}$ $\tau_{2\,3}$ $\tau_{2\,5}$ $\tau_{3\,2}$ $\tau_{3\,4}$ $\tau_{3\,6}$ $\tau_{3\,8}$ $\tau_{4\,1}$ $\tau_{4\,3}$ $\tau_{4\,5}$ $\tau_{5\,2}$ $\tau_{5\,4}$ $\tau_{5\,6}$ $\tau_{5\,8}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{6\,3}$ $\tau_{6\,5}$ $\tau_{7\,2}$ $\tau_{7\,4}$ $\tau_{7\,6}$ $\tau_{7\,8}$ $\tau_{8\,1}$ $\tau_{8\,3}$ $\tau_{8\,5}$ $\tau_{8\,7}$ $\tau_{9\,2}$ $\tau_{9\,4}$ $\tau_{9\,6}$ $\tau_{9\,8}$ $\tau_{10\,1}$ $\tau_{10\,3}$ $\tau_{10\,5}$ $\tau_{10\,7}$								
4	$\tau_{1\,1}$ $\tau_{1\,5}$ $\tau_{2\,2}$ $\tau_{3\,3}$ $\tau_{4\,1}$ $\tau_{5\,1}$ $\tau_{5\,5}$ $\tau_{6\,4}$ $\tau_{7\,3}$ $\tau_{7\,8}$ $\tau_{8\,2}$ $\tau_{8\,5}$ $\tau_{9\,1}$ $\tau_{10\,4}$ $\tau_{10\,5}$	$\tau_{1\,2}$ $\tau_{1\,6}$ $\tau_{2\,1}$ $\tau_{2\,5}$ $\tau_{3\,2}$ $\tau_{3\,4}$ $\tau_{3\,8}$ $\tau_{4\,2}$ $\tau_{4\,3}$ $\tau_{5\,2}$ $\tau_{5\,6}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{6\,5}$ $\tau_{7\,4}$ $\tau_{7\,7}$ $\tau_{8\,3}$ $\tau_{8\,6}$ $\tau_{9\,2}$ $\tau_{9\,6}$ $\tau_{10\,1}$ $\tau_{10\,6}$	$\tau_{1\,3}$ $\tau_{1\,7}$ $\tau_{2\,4}$ $\tau_{3\,1}$ $\tau_{3\,5}$ $\tau_{3\,7}$ $\tau_{4\,4}$ $\tau_{5\,3}$ $\tau_{5\,7}$ $\tau_{5\,9}$ $\tau_{6\,2}$ $\tau_{7\,1}$$\tau_{7\,5}$ $\tau_{8\,4}$ $\tau_{8\,7}$ $\tau_{9\,3}$ $\tau_{9\,7}$ $\tau_{9\,8}$ $\tau_{10\,2}$ $\tau_{10\,7}$	$\tau_{1\,4}$ $\tau_{2\,3}$ $\tau_{3\,2}$ $\tau_{3\,6}$ $\tau_{4\,5}$ $\tau_{5\,8}$ $\tau_{6\,3}$ $\tau_{7\,6}$ $\tau_{8\,1}$ $\tau_{9\,4}$ $\tau_{9\,5}$ $\tau_{10\,3}$						
8	$\tau_{1\,1}$ $\tau_{3\,7}$ $\tau_{5\,5}$ $\tau_{6\,4}$ $\tau_{7\,3}$ $\tau_{9\,1}$	$\tau_{1\,2}$ $\tau_{2\,1}$ $\tau_{3\,8}$ $\tau_{4\,3}$ $\tau_{5\,6}$ $\tau_{7\,4}$ $\tau_{8\,3}$ $\tau_{9\,2}$ $\tau_{10\,1}$ $\tau_{6\,5}$	$\tau_{1\,3}$ $\tau_{2\,2}$ $\tau_{3\,1}$ $\tau_{4\,4}$ $\tau_{5\,7}$ $\tau_{7\,5}$ $\tau_{8\,4}$ $\tau_{9\,3}$ $\tau_{10\,2}$	$\tau_{1\,4}$ $\tau_{2\,3}$ $\tau_{3\,2}$ $\tau_{4\,1}$ $\tau_{5\,8}$ $\tau_{7\,6}$ $\tau_{8\,5}$ $\tau_{9\,4}$ $\tau_{10\,3}$	$\tau_{1\,5}$ $\tau_{3\,3}$ $\tau_{4\,5}$ $\tau_{5\,1}$ $\tau_{5\,9}$ $\tau_{7\,7}$ $\tau_{8\,6}$ $\tau_{9\,5}$ $\tau_{10\,4}$	$\tau_{1\,6}$ $\tau_{2\,5}$ $\tau_{3\,4}$ $\tau_{5\,2}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{7\,8}$ $\tau_{9\,6}$ $\tau_{10\,5}$	$\tau_{1\,7}$ $\tau_{2\,4}$ $\tau_{3\,5}$ $\tau_{5\,3}$ $\tau_{6\,2}$ $\tau_{7\,1}$ $\tau_{8\,7}$ $\tau_{9\,7}$ $\tau_{10\,6}$	$\tau_{3\,6}$ $\tau_{4\,2}$ $\tau_{5\,4}$ $\tau_{6\,3}$ $\tau_{7\,2}$ $\tau_{8\,1}$ $\tau_{8\,2}$ $\tau_{9\,8}$ $\tau_{10\,7}$		
10	$\tau_{1\,1}$ $\tau_{4\,2}$ $\tau_{5\,7}$ $\tau_{6\,1}$ $\tau_{7\,5}$ $\tau_{8\,4}$ $\tau_{9\,3}$	$\tau_{1\,2}$ $\tau_{2\,1}$ $\tau_{4\,3}$ $\tau_{10\,1}$ $\tau_{5\,8}$ $\tau_{7\,6}$ $\tau_{8\,5}$ $\tau_{9\,4}$	$\tau_{1\,3}$ $\tau_{3\,1}$ $\tau_{4\,4}$ $\tau_{10\,2}$ $\tau_{7\,7}$ $\tau_{8\,6}$ $\tau_{9\,5}$	$\tau_{1\,4}$ $\tau_{2\,3}$ $\tau_{3\,2}$ $\tau_{4\,1}$ $\tau_{10\,3}$ $\tau_{5\,9}$ $\tau_{7\,8}$ $\tau_{8\,7}$ $\tau_{9\,6}$	$\tau_{1\,5}$ $\tau_{3\,3}$ $\tau_{5\,1}$ $\tau_{10\,4}$ $\tau_{9\,7}$	$\tau_{1\,6}$ $\tau_{2\,5}$ $\tau_{3\,4}$ $\tau_{5\,2}$ $\tau_{5\,10}$ $\tau_{10\,5}$ $\tau_{9\,8}$	$\tau_{1\,7}$ $\tau_{2\,4}$ $\tau_{3\,5}$ $\tau_{5\,3}$ $\tau_{6\,2}$ $\tau_{7\,1}$ $\tau_{10\,6}$ $\tau_{2\,2}$	$\tau_{3\,6}$ $\tau_{5\,4}$ $\tau_{6\,3}$ $\tau_{7\,2}$ $\tau_{8\,1}$ $\tau_{8\,2}$ $\tau_{10\,7}$ $\tau_{7\,3}$ $\tau_{9\,1}$	$\tau_{3\,7}$ $\tau_{5\,5}$ $\tau_{6\,4}$	$\tau_{3\,8}$ $\tau_{4\,5}$ $\tau_{5\,6}$ $\tau_{6\,5}$ $\tau_{7\,4}$ $\tau_{8\,3}$ $\tau_{9\,2}$

Tab. 5.5: Step-to-processor mapping of the synthetic e2e flows

```
tg_cnt 10
task_cnt 7 3
task_degree 3 3
period_mul 1,0.85,1.02
tg_write
eps_write

table_cnt 1
table_label Processor
type_attrib WCET 10 10
trans_write
```

Listing 5.1: TGFF input code

In order to determine which of the proposed algorithms shows the best performance, the term Relative Breakdown Utilization (RBU) for time-partitioned systems is

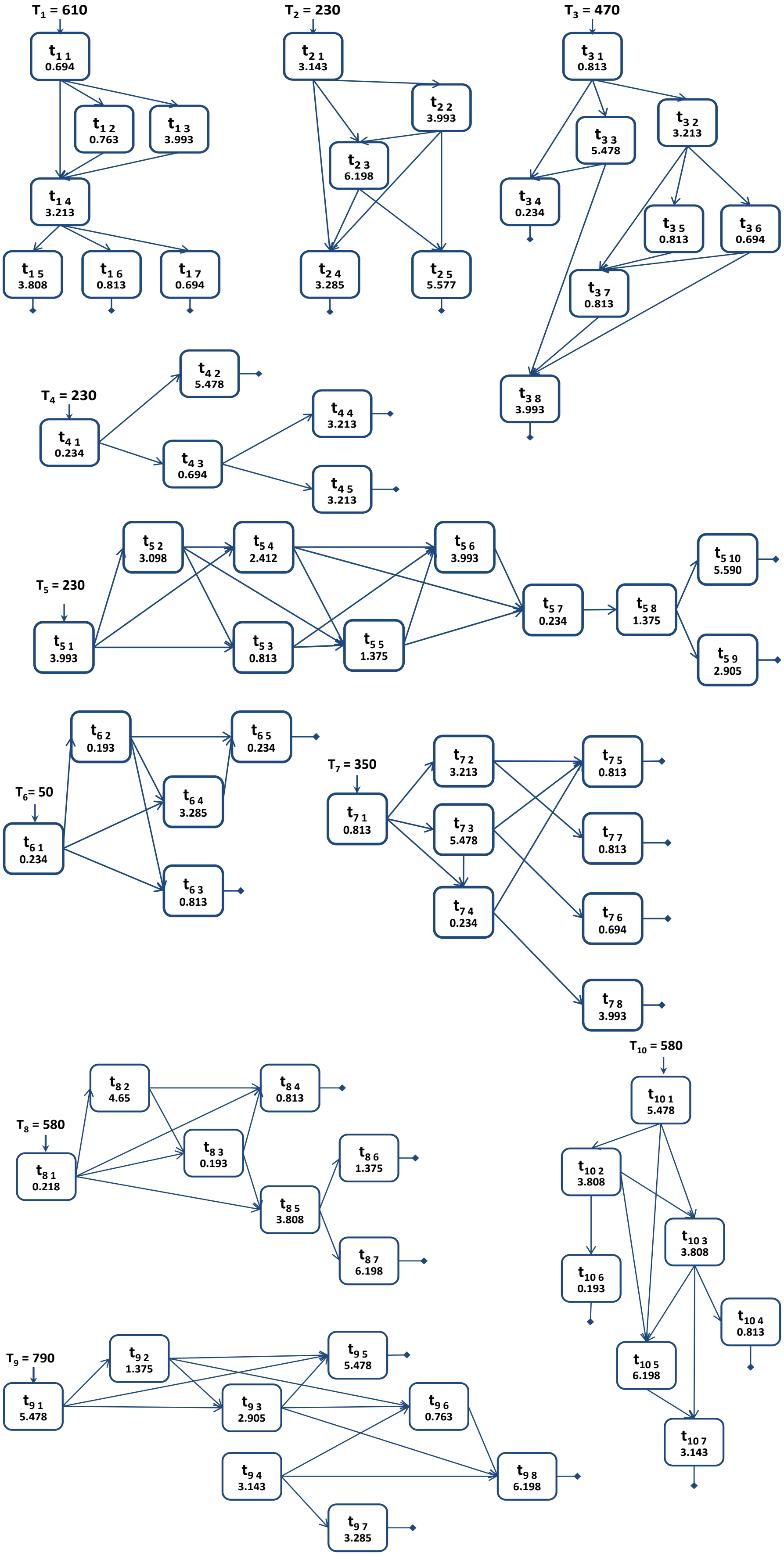


Fig. 5.2: Synthetic e2e flows

introduced. It is based on the Breakdown Utilization (BU) introduced in [LSD89], extended for application to partitioned systems where CPU time is not fully available for a partition. Thus, the RBU term (in percentage) for a partition is the value calculated as $U_{P_x}/AU_{P_x} * 100$, reached when all the execution times are scaled up to a point at which a deadline is first missed. To do so, a scaling factor is applied to all execution times in the context of each partition, until the system reaches the boundary of schedulability. For those tests with more than one processor, we will take the average RBU to show the experimental results, which is possible because step-to-processor mapping is kept fairly balanced in all cases. As a reference, a non-partitioned system where the CPU is fully available (general FP systems) will be considered. Thus, all the experiments considering a single partition with $AU = 100\%$ will be replicated, so the BU for each algorithm will be shown.

When deadlines are too restrictive or the system is barely schedulable, and hence for calculating the RBU/BU the execution times of tasks have to be drastically reduced, it is considered that when partitions reach a utilization lower than 1%, the system is not schedulable (NS). This happens because the response time analysis computes all the time-window gaps where the execution is not allowed and thus worst-case response-times increase. Proposing an optimized partition window assignment is beyond the scope of the work presented in this chapter, and it will be addressed in Chapter 6.

	$D = T$	$D = 2T$	$D = 4T$	$D = 6T$	$D = 8T$	$D = 10T$
1 Processor	EQS (54)	EQS (75)	EQS (89)	EQS (93)	EQS (95)	EQS (97)
2 Processors	NS -	NS -	EQS (83)	EQS (88)	EQS (90)	EQS (93)
4 Processors	NS -	NS -	EQS PD_Local NPD_Local (66)	EQS,PD_Global PD_Local,NPD_Local (71)	EQS (76)	EQS,PD_Local NPD_Local PD_Global, NPD_Global (76)
8 Processors	NS	NS - -	PD_Local (66)	NPD_Local (76)	EQS,PD_Local NPD_Local (76)	EQS,PD_Local NPD_Local NPD_Global (76)
10 Processors	NS -	NS -	NPD_Local (62)	EQS,PD_Local, NPD_Local NPD_Global (62)	EQF, EQF, PD_Local PD_Global, NPD_Local NPD_Global (62)	NPD_Local (75)

Tab. 5.6: RBU results (in %) for each algorithm (with partitioning)

The most relevant results have been compiled in Tables 5.6 and 5.7. They show the algorithm or algorithms that obtain the highest RBU/BU (expressed in % in brackets) for each combination of number of processors and deadline/period rates. Generally comparing Tables 5.6 and 5.7, we can directly see the penalisation that occurs when 100% of CPU-time is not available, i.e. without time partitioning,

	$D = T$	$D = 2T$	$D = 4T$	$D = 6T$	$D = 8T$	$D = 10T$
1 Processor	NPD_Local (63)	EQS (78)	EQS (89)	EQS (93)	EQS (95)	EQS (96)
2 Processors	PD_Local, NPD_Local (61)	EQS (76)	EQS (89)	EQS (93)	EQS (94)	EQS (95)
4 Processors	PD_Local, NPD_Local (57)	PD_Local (72)	EQS (85)	EQS (91)	EQS (93)	EQS (94)
8 Processors	PD_Local, PD_Global NPD_Local (49)	EQS, PD_Local, NPD_Local (64)	EQS, PD_Local NPD_Local NPD_Global (78)	EQS PD_Local NPD_Local (84)	EQS (88)	EQS (96)
10 Processors	NPD_Local (46)	EQS (63)	NPD_Local (80)	EQS (86)	EQS (89)	EQS (91)

Tab. 5.7: BU results (in %) for each algorithm

those scenarios where D=T are schedulable, whereas with time partitioning only the scenario with a single processor is schedulable. Even if the RBU term is relative to the CPU availability, the CPU time not available for the partition and the effect it has on the analysis technique for time-partitioned systems, provokes that utilizations in all scenarios are always lower than their counterparts in non-partitioned ones.

Regarding Table 5.6, which corresponds to time-partitioned systems, when the computing is performed in a single processor, the algorithm that obtains the highest RBU is always EQS. When the number of processors is increased some other algorithms seem to behave better, such as NPD_Local that produces the highest RBU when deadlines are 6, 8 and 10 times the period. Moreover, PD_Local also exhibits a high performance in distributed systems where deadlines are 4, 6 or 8 times the period. In systems without time-partitioning (Table 5.7) NPD_Local always obtains the highest BU when deadlines are within the periods, and in this scenario PD_Local also exhibits good performance when steps are mapped in more than one processor. When deadlines tend to relax, EQS is the most suitable in most of the experiment-configurations. Note that the UD and ED algorithms, which obtain fairly good results in the simple example from Section 5.2, never appear in any of these tables as the best ones.

In order to complete the qualitative analysis of these tests, Figures 5.3 to 5.7 show, for each processor, the evolution of the RBU obtained by the proposed algorithms as a function of the D/T ratio. This allows readers to track the performance of those algorithms that are not present in the tables, as in those tables only the outstanding algorithms in each experiment configuration are presented.

As a general conclusion, it can be stated that there is no single algorithm that behaves the best in all cases. This reinforces the idea of proposing a set of simple and non-iterative algorithms so that all of them can be applied, and then the most

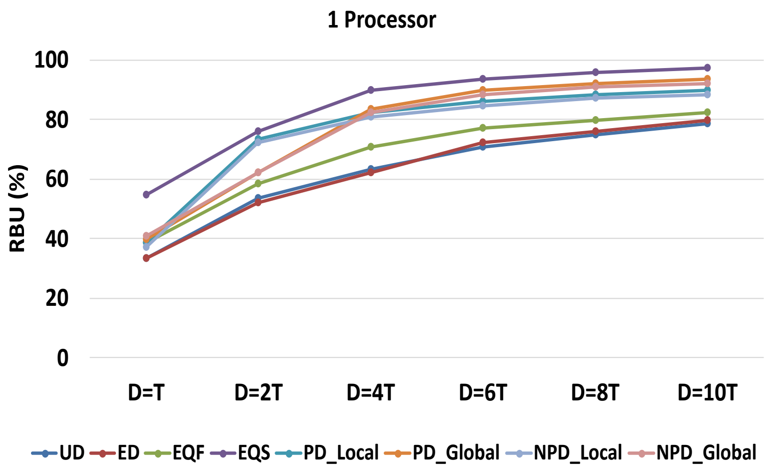


Fig. 5.3: RBU evolution in 1 processor

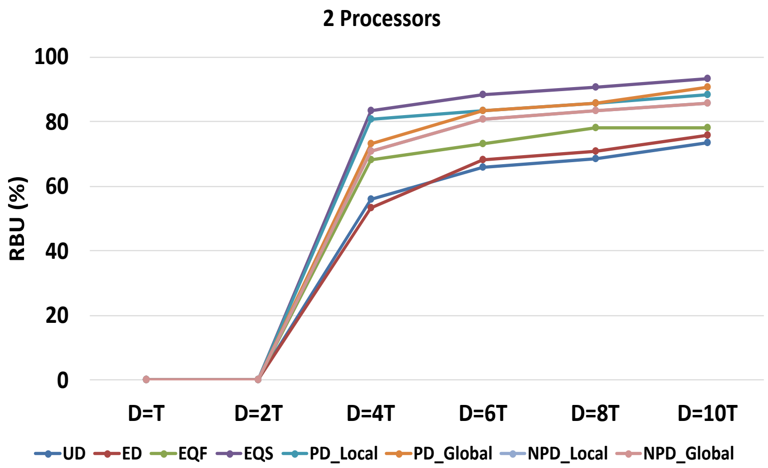


Fig. 5.4: RBU evolution in 2 processors

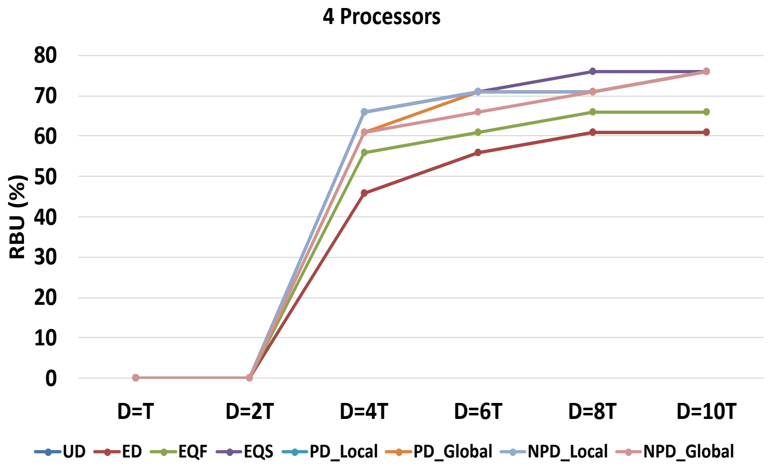


Fig. 5.5: RBU evolution in 4 processors

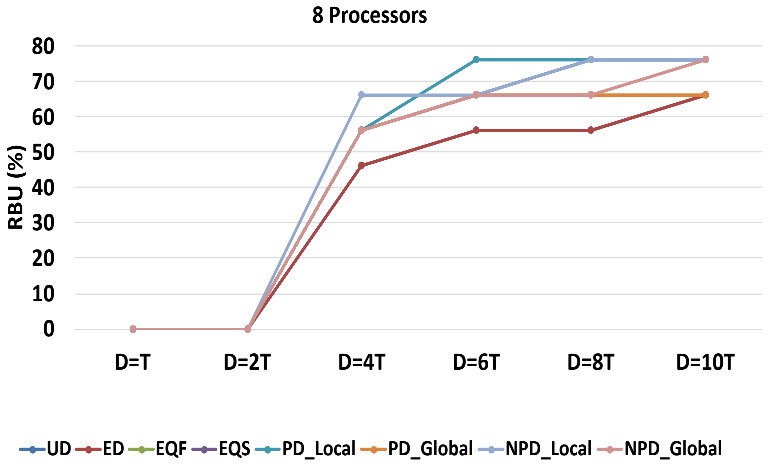


Fig. 5.6: RBU evolution in 8 processors

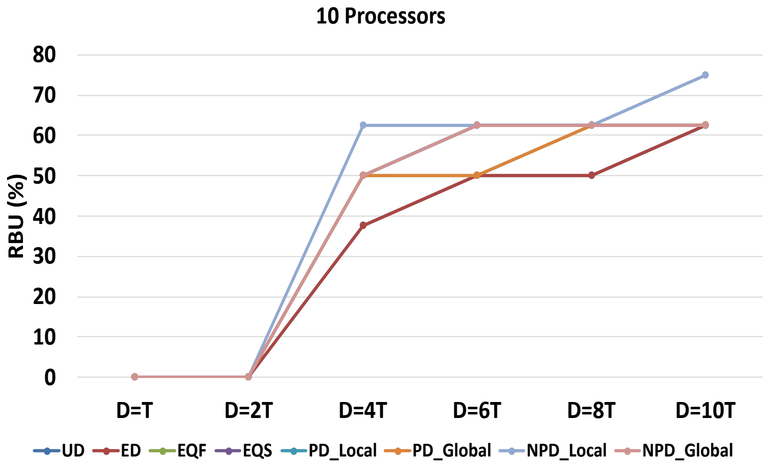


Fig. 5.7: RBU evolution in 10 processors

Experiment	Execution time (s)
Simple example	26
Railway signalling use-case	28
1 Processor - D=T	36
1 Processor - D=10T	27
10 Processors - D=10T	31

Tab. 5.8: Execution time of some of the experiments

suitable result can be selected. Table 5.8 shows the execution times of some of the experiments conducted in this work: the simple example from Section 5.2.1, the railway signalling use-case from Section 5.3.1 and a subset of the synthetic scenarios from Section 5.3.2. In each experiment, the execution time shown includes the execution of the eight priority assignment algorithms plus their worst-case response-time analysis.

5.4 Conclusions

In this chapter, a collection of algorithms for priority assignment of multipath e2e flows in hierarchically scheduled and time-partitioned distributed real-time systems has been presented. All of them are non-iterative algorithms that can provide feasible solutions in a short time, in contrast to iterative algorithms that might require long computation times, so designers and developers have the possibility of applying all of them in order to check which one best suits the target system. Results show that there is no single algorithm that stands out clearly from the others. Therefore, the algorithm that best suits a certain configuration should be chosen from this collection, although an insight into the behavior can be obtained by comparing the targeted scenario with the ones evaluated in this work. The proposed algorithms have been applied to the target industrial use-case, then they have been evaluated and their performance has been ranked by means of a synthetic representative application.

6 Partition window assignment

Partitioning techniques are implemented in the development of safety-critical applications to ensure isolation among components, as explained in the industrial use-case description in Chapter 1. As identified in the literature review (Chapter 2), there are few research works addressing the scheduling optimization of time-partitioned real-time systems and they are fairly recent.

An adequate arrangement of the execution of such partitions is a key challenge so that applications meet the imposed hard deadlines. In this chapter, the effect of window sizes and context switch overheads in the partition window configuration is studied, with the aim of analyzing their impact when the response time analysis and priority assignment techniques are applied. Then, a heuristic algorithm is proposed, in order to obtain a partition window configuration that enables the schedulability of partition-based safety critical systems. This algorithm is evaluated in synthetic test scenarios and it is also applied to the industrial use-case.

6.1 Study of the influence of partition windows on schedulability

To perform the proposed study, different partition scheduling schemes will be evaluated. Thus, a simple application example is constructed, which includes the most relevant features that characterize the motivating railway use-case. This example is composed of a single multipath e2e flow activated periodically every 50 ms which is composed of six steps ($\tau_{1\ 1}$ to $\tau_{1\ 6}$), as shown in Figure 6.1. The e2e flow is mapped within a single partition (P_x) composed of a single partition window, and it is assumed, for the sake of simplicity, that the worst-case execution time of each step is fixed and equal to 2 ms. The priority of each step is shown in brackets, and the MAF considered for the whole experiment set is 50 ms. When referring to the schedulability of the application, it relates to the worst-case response time of $\tau_{1\ 6}$ ($R_{1\ 6}$) in comparison with its deadline.

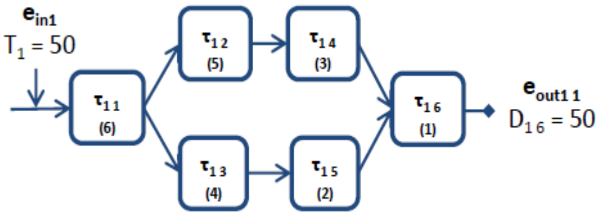


Fig. 6.1: Guiding application example

6.1.1 Available Utilization

A very common early-design decision regards to the CPU time allocated for the execution of each partition, i.e. AU_{P_x} in the example described here. Depending on this time, response times may vary significantly as shown in Figure 6.2. Even if this effect seems obvious, it gives us an idea about the effect that not having all the processor time dedicated for the execution of applications produces on the worst-case response times calculated by the analysis technique. For a fixed MAF and considering a fixed number of windows, a lower available utilization generates longer gaps between partition windows, thus the longer these gaps are (where P_x is not allowed to be executed), the higher is the worst-case response time. In this example the partition utilization U_{P_x} represents 24% of the CPU time, and worst-case response times vary from 580 ms to 12 ms when the available CPU goes from the initial value of 24% to 100% respectively.

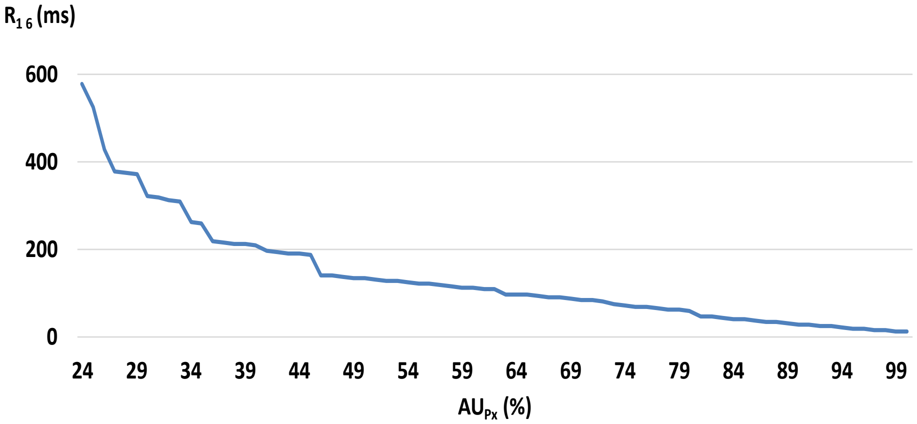


Fig. 6.2: Worst-case response time of $\tau_{1\ 6}$ as a function of AU_{Px}

6.1.2 Number of windows

Once the available time has been fixed for a partition, the next design decision to take is to distribute this given time in the MAF. At a first approach, a uniform distribution of partition windows will be considered, although this might be subject to optimization when more partitions make up the MAF. Figure 6.3 shows the worst-case response times obtained for this example when the number of partition windows varies from 1 to 100, for three different values of AU_{P_x}. As can be seen, increasing the number of windows produces a reduction in the length of the unavailable gaps in the MAF, leading to a remarkable reduction of the obtained response times. Again, we can observe here how having higher gaps between windows leads to higher response times.

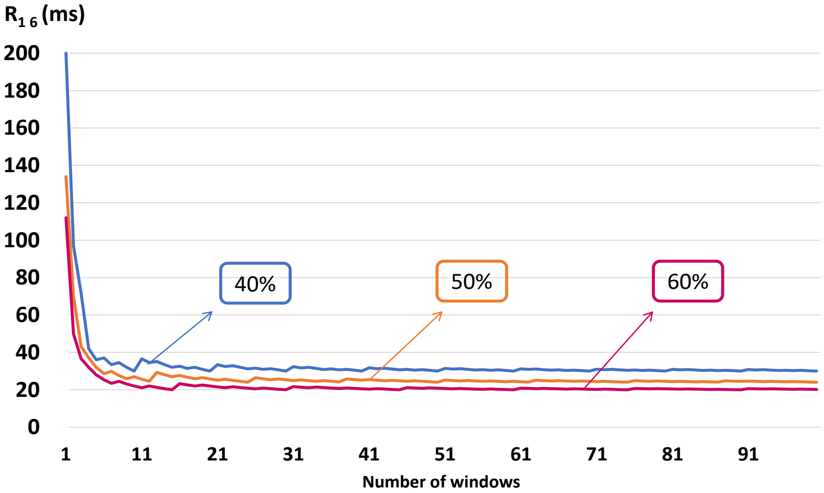


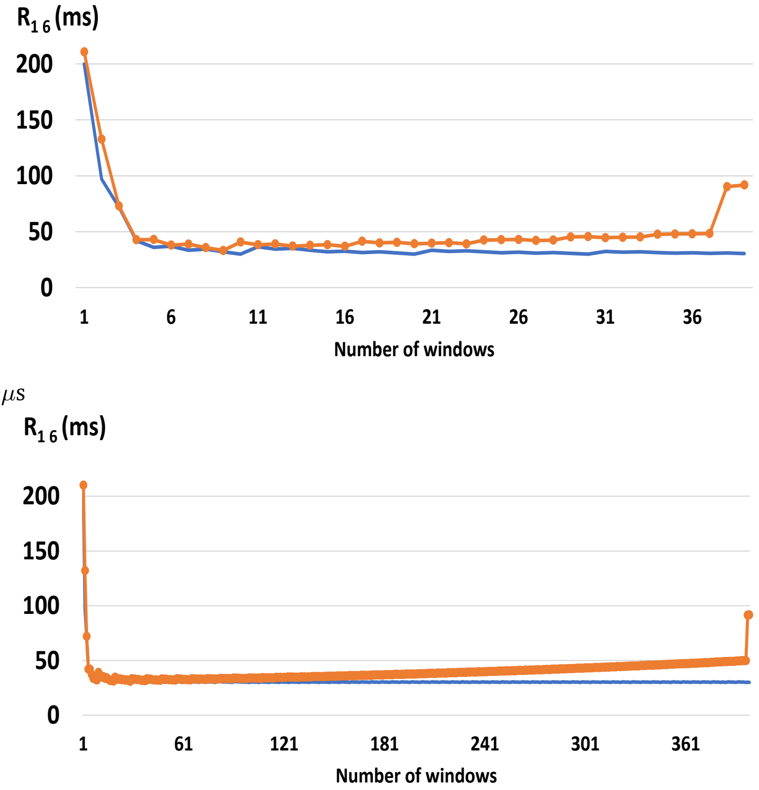
Fig. 6.3: Evolution of worst-case response time of $\tau_{1\ 6}$ with the number of partition windows for different values of AU_{P_x} (in %)

6.1.3 Context switch overheads

Experiments conducted until now have not considered the effects of the context switch overheads that are present when a partition is activated, and which have been modeled in Section 3.3. In general, context switch overheads depend on the operating system/hypervisor where applications are executed. For instance, in [Ham+] the measured context switch overhead in a hypervisor is 17μs, and in [Mas+], it is 27μs. Therefore, in the experiments performed in this thesis, a value of $CS = 20\mu$s is considered, along with a higher order of magnitude representing a slower processor, i.e. $CS = 200\mu$s. Formally, the maximum CPU time that can be spent in context switch overheads in processor CPU_y, is bounded by the difference between the available partition utilization (AU_{P_x}) and the partition utilization (U_{P_x}). Considering this, the limit of the number of partition windows that can be set for partition P_x without overloading the partition is defined as follows:

$$NW_{P_x} = \lfloor (AU_{P_x} - U_{P_x}) * \frac{MAF_y}{CS_y} \rfloor \tag{6.1}$$

In other words, with the number obtained by this expression or a lower value of windows, enough time for the execution of partition P_x is guaranteed.

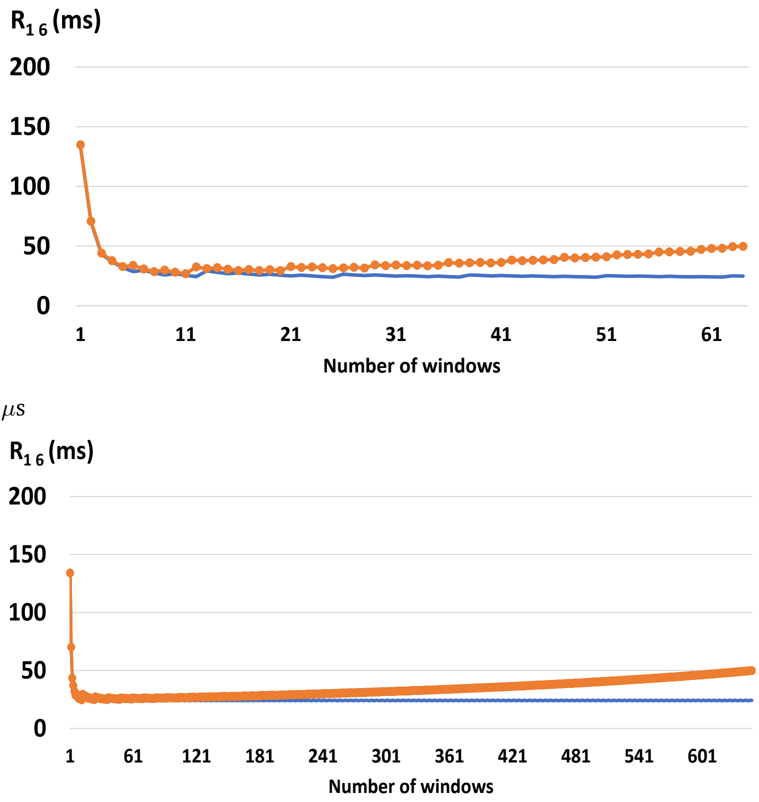


(a) $CS = 200\mu s$

(b) $CS = 20\mu s$

Fig. 6.4: Worst-case response time as a function of the number of partition windows - $AU_{P_x} = 40\%$

Figures 6.4 to 6.6 show the worst-case response times obtained for different numbers of partition windows, available utilizations and context switch overhead values for this guiding example. For different AU_{P_x} values we calculate the response times when increasing the number of partition windows in the range 1 to NW_{P_x}. Qualitatively, worst-case response times vary in the same way regardless of the CPU availability, i.e. the maximum response times are obtained when P_x is scheduled in a single window, and as the number of windows increases response times reduce fast, up to a point. After that point, notice that in the ideal scenario, $CS_y = 0$ (blue plot), the response-time curve remains constant, while if $CS_y > 0$ (orange plot) the curve increases again.



(a) $CS = 200\mu s$

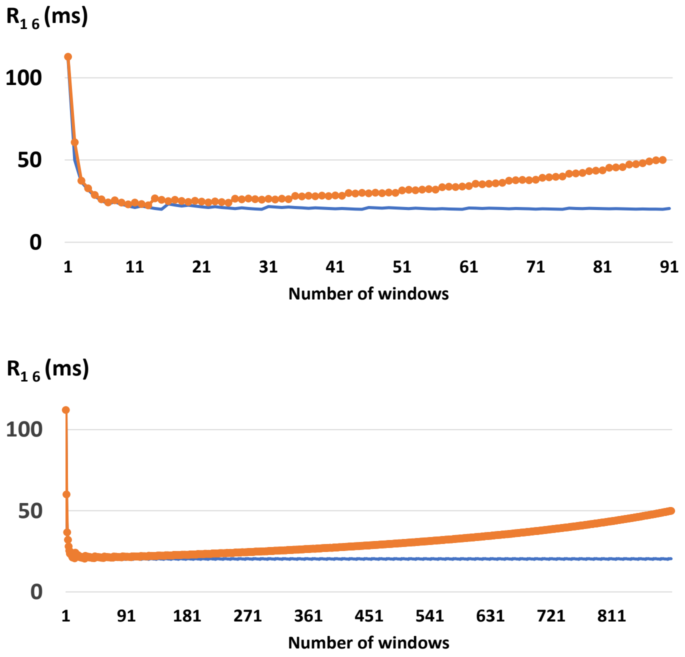
(b) $CS = 20\mu s$

Fig. 6.5: Worst-case response time as a function of the number of partition windows - $AU_{P_x} = 50\%$

6.1.4 Conclusions of the study

It has been observed that increasing the number of partition windows has a positive effect on the reduction of response times, which is outweighed by the negative effect of context switch overheads. Finding the window configuration that produces the turning point in response times is essential for a partition window optimization algorithm.

When the partition utilization is very low in comparison to the available utilization, and also when context switch overheads are very low, the range of windows to explore ($1..NW_x$ for P_x) is very large. It has been found that analyzing a very big window number ($NW_x > 500$) has a negative impact on the execution time of the analysis tool. Therefore, the strategy to follow will be to start to explore from minimum number of windows and then increase the number until the turning point is found.



(a) $CS = 200\mu s$

(b) $CS = 20\mu s$

Fig. 6.6: Worst-case response time as a function of the number of partition windows - $AU_{P_x} = 60\%$

6.2 Heuristic partition window assignment

In the previous study, the effect of the configuration parameters (referred to the available utilization and the number of windows within the MAF) on the worst-case response times has been analyzed. Now, an algorithm called Window Assignment (WinAs) is proposed in section 6.3.1 in order to leverage this knowledge, by searching for the number of windows that produces a schedulable solution for a fixed available utilization in each partition. WinAs is then integrated within an outer algorithm, which starts from a given available utilization for each partition. This outer algorithm, called Heuristic Optimized Partition Window Assignment (HOPWA) executes WinAs iteratively, as will be shown later in section 6.3.2, with the aim of providing schedulable solutions while the available utilization of each partition is optimized.

6.2.1 Window Assignment (WinAs) Algorithm

The WinAs algorithm tries to find a schedulable system configuration by increasing the number of windows for a fixed available utilization assigned to each partition. It produces, in each processor, an adjusted MAF where there is one partition window for each partition, as well as a priority assignment. This algorithm exploits the results of the study in Section 6.2, where the influence of the number of partition windows within the MAF and the gap between these windows on the response is shown.

The design of the algorithm is described in Algorithm 8. The rationale for this design is to vary the number of partition windows, by always having a single window per partition within a diminishing MAF. Reducing the MAF is equivalent to increasing the number of windows, assuming a uniform window distribution, within a fixed MAF. For example: considering a 100 ms MAF, the window for a certain partition P_1 whose AU_{P1} is 40% will first be defined as follows: $Win_{11} = \{0, 40\}$. In order to increase the number of windows (assuming that the solution was not schedulable), WinAs sets the new MAF in that processor to 50 ms. Therefore, the new partition window will be defined as: $Win_{11} = \{0, 20\}$, which is equivalent to dividing the previous 40 ms window into two uniformly distributed 20 ms windows within the prior 100 ms MAF. Performing a non-uniform window assignment remains as future research work, bearing in mind that, as noticed in the study from Section 6.2, worst-case response times are highly influenced by the longest gap between windows within the MAF.

Algorithm 8: Window Assignment (WinAs) algorithm

```
for each $CPU_y$ do
  $MAF_y = min(D_{ij} \in CPU_y)$
end for
while Stopping criterion not met do
  for each $CPU_y$ do
    $Offset = 0$
    for each $P_x$ in $CPU_y$ do
      $Win_{x1} = \{Offset, L_{x1}\}$
      $Offset = Offset + L_{x1}$
    end for
  end for
  Perform Priority assignment
  if Schedulable then
    Return SUCCESS
  else
    for each $CPU_y$ do
      $MAF_y = MAF_y / Q$
    end for
  end if
end while
Return FAIL
```

The first step of the algorithm consists of calculating the initial MAF values (line 2). It is calculated, in the context of each processor, as follows: the value of the MAF will be the most restrictive deadline (the lowest value) of all the steps present in that processor. In each iteration, a single window is defined for each partition within the MAF (line 8), its length being proportional to the available utilization of that partition, which for partition P_x is calculated as follows:

$$L_{x1} = MAF * AU_{Px} \tag{6.2}$$

Since the schedulability analysis of each partition is performed independently (see Chapter 4), the worst-case response times do not depend on a specific partition ordering within the MAF. Therefore, the newly defined partition windows (one per partition) are placed one after another in an arbitrary ordering, based on the partition's index.

As shown in Chapter 5, priorities assigned to the steps within time partitions have a big impact on the system's schedulability. In this step (line 12), all the algorithms described in Chapter 5 are evaluated, and the best solution is selected at this stage. To determine the best priority assignment, a figure of merit is proposed for schedulable solutions: for each e2e flow the maximum value R_{ij}/D_{ij} among all output steps is calculated, so that the worst result per e2e flow is captured; then, the average R_{ij}/D_{ij} ratio of all e2e flows is calculated, so the algorithm's result that obtains the lowest average ratio is considered the best solution.

If the system is schedulable, the algorithm succeeds (line 14), whereas if it is not, the MAF is reduced by an adjustable factor (Q in line 17), which should be greater than 1, and Eq.6.2 is applied to the single window of each partition in order to maintain the appropriate available utilization. In the previous example, where the rationale of this algorithm has been explained, the reduction factor applied to the MAF was $Q = 2$, as the MAF was reduced from 100 ms to 50 ms. This value will also be used for the performance evaluation in the next section. The adjustment of factor Q constitutes an optimization problem in itself, and it may be addressed in future work.

The stopping condition evaluated in line 4 of Algortithm 8 will be met when all partitions reach the maximum number of windows, which can be calculated directly by Eq. 6.1. If during this search a schedulable solution is not found, WinAs fails (line 21), and returns the MAF value that produces the highest SSF value among all the explored values, where a single partition window is assigned to each partition according to its available utilization.

6.2.2 Heuristic Optimized Partition Window Assignment (HOPWA)

The development of WinAs, which searches for a schedulable solution having a fixed available utilization in each partition, enables the design of a new algorithm based on this, which can explore different available utilizations of partitions in order to optimize these values in the context of the whole system. This algorithm, called Heuristic Optimized Partition Window Assignment (HOPWA) receives an initial available utilization for each partition, and it tries to optimize it when a schedulable solution is found. In the study in Section 6.2, it was concluded that increasing the available utilization of partitions reduces the worst-case response times, so if a schedulable solution is not found from the initial available utilization values, a method to distribute the processors' spare utilization (SU) is used to try to reach schedulability by increasing the partitions' available utilization.

Thus, HOPWA's inputs are: a step-to-partition allocation, a partition-to-processor allocation and a given initial available utilization for each partition. This initial available utilization, AU_{Px}^{init}, should be at least equal to the partition utilization (U_{P_x}), and the total available utilization of all the partitions allocated to a processor must not be higher than 1. Then, the spare utilization in each processor (SU_{CPU_y}), is directly calculated by subtracting from 1, i.e. from the whole processing time, the sum of all the AU_{Px} of the hosted partitions. This produces an optimized available utilization for each partition, which hosts a single partition window within an adjusted MAF. As it executes WinAS iteratively, a priority assignment is also produced.

HOPWA is described by its pseudo-code in Algorithm 9. As will be shown later, a binary search strategy will be used to tune the available utilization of each partition. This binary search will be conducted between a minimum and a maximum available utilization, AU_{Px}^{min} and AU_{Px}^{max} respectively, where AU_{Px}^{mid} will be the mid point between them.

In the initialization phase of the algorithm each partition's available utilization is set to the initial value AU_{Px}^{init} , as can be seen in line 2. For each partition, AU_{Pi}^{min} is set to 0 in order to enable the optimization of the available utilization later on, and AU_{Pi}^{max} is set to 1. Then, WinAs is executed for the first time, with the current available utilization of each partition as an input. Two situations may take place, but in both of them the following step consists of tuning the available utilization of all partitions, be it to reach schedulability or to optimize the available utilization assigned to each partition:

Algorithm 9: Heuristic Optimized Partition Window Assignment (HOPWA)

```
**Initialization:**
$\forall P_x \leftarrow AU_{Px} = AU_{Px}^{init}, AU_{Px}^{max} = 1, AU_{Px}^{min} = 0$

Execute **WinAs($\forall AU_{Px}$)**

**if** **WinAs** returns FAIL **then**
  Distribute $SU_{CPU_y}$ according to Eq.6.3
  Execute **WinAs** **($\forall AU_{Px}$)**
  **if** **WinAs** returns FAIL **then**
    Return FAIL
  **else**
    **for** each $P_x$ **do**
      $AU_{Px}^{min} = AU_{Px}^{init}$
      $AU_{Px}^{max} = AU_{Px}$
      $AU_{Px}^{mid} = mid(AU_{Px}^{min}, AU_{Px}^{max})$
      $AU_{Px} = AU_{Px}^{mid}$
    **end for**
  **end if**
**else**
  **for** each $P_x$ **do**
    $AU_{Px}^{max} = AU_{Px}$
    $AU_{Px}^{mid} = mid(AU_{Px}^{min}, AU_{Px}^{max})$
    $AU_{Px} = AU_{Px}^{mid}$
  **end for**
**end if**

**while** Stopping criterion not met **do**
  Execute **WinAs($\forall AU_{Px}$)**
  **if** WinAs returns SUCCESS **then**
    **for** each $P_x$ **do**
      $AU_{Px}^{max} = AU_{Px}$
      $AU_{Px}^{mid} = mid(AU_{Px}^{min}, AU_{Px}^{max})$
      $AU_{Px} = AU_{Px}^{mid}$
    **end for**
  **else**
    **for** each $P_x$ **do**
      $AU_{Px}^{min} = AU_{Px}$
      $AU_{Px}^{mid} = mid(AU_{Px}^{min}, AU_{Px}^{max})$
      $AU_{Px} = AU_{Px}^{mid}$
    **end for**
  **end if**
**end while**
Return SUCCESS
```

1. If WinAs does not find a schedulable solution (line 6), the spare utilization of each processor is distributed among all the partitions uniformly, and added to each partition's available utilization (Line 7), as expressed by the following equation:

   $$AU_{Px} = AU_{Px} + SU_{CPU_y}/NP_{CPU_y} \tag{6.3}$$

   where NP_{CPU_y} is the number of partitions in CPU_y.

   Then, WinAs is executed again (line 8). If there is a schedulable solution (line 11), the search should be conducted, for each partition, in the range of the available utilization ($AU_{Px}^{min} = AU_{Px}^{init}$) initially assigned and the current available utilization assigned after distributing the processors' available utilization ($AU_{Px}^{max} = AU_{Px}$). This is performed following a binary search strategy for the available utilization of each partition, as shown in lines 27-42. WinAs is executed at each iteration in order to find a schedulable solution. The stopping condition of the algorithm will be, for all cases henceforth, a precision value in the variations of AU_{Px}, which may be adjustable.

   On the contrary, if WinAs fails in finding a schedulable solution right after distributing the processors' spare utilization (line 9), HOPWA fails. With the aim of providing a solution, although not schedulable, the MAF values explored during the execution of WinAs in this stage are stored in memory and the System Slack Factor (SSF), defined in Section 3.4, is calculated for each solution. The MAF value that produces the highest SSF is kept as the solution returned by the algorithm.

2. If WinAs finds a schedulable solution (Line 19), the upper bound for the binary search conducted in lines 27-42 is set to the initial available utilization (Line 21). After the previously stated stopping condition is met, the algorithm succeeds.

6.3 Performance evaluation

In this section the algorithms proposed in Section 6.3 are evaluated. First, the WinAs algorithm is characterized through a set of synthetic experiments. Then, HOPWA is applied to the same synthetic scenarios, as well as to the motivational railway use-case.

6.3.1 Design of the synthetic experiments

As seen in the study in Section 6.2, varying the number of windows within the MAF may be beneficial for some e2e flow's response times and detrimental for others, since the algorithm tries to schedule the most restrictive e2e flows first by selecting an appropriate MAF for them. Therefore, the least restrictive ones suffer an increase in their response times, due to the large context switch overheads. As the study has been performed for a single e2e flow allocated to a single partition, it is necessary to analyze the behavior of WinAs, in order to assess its capacity to search for schedulable solutions when different e2e flows, in terms of deadline requirements and/or load, are part of the same target system. Moreover, the effect of considering some e2e flows that are allocated only to a subset of the processors needs to be evaluated, since, as shown in the description of the algorithm, the solution space may be different in the context of each processor.

With all these features in mind, a set of synthetic scenarios will be evaluated. To do so, a baseline synthetic scenario, referenced as **Scn1** from now on and depicted in Figure 6.7, has been designed. It is composed of four e2e flows with different extensive deadline requirements, and they are allocated to four processors. Then, some modifications upon this base scenario will be performed, which include modifying the step-to-processor allocation in **Scn2** and **Scn3** (Figures 6.8 and 6.9 respectively) and also modifying their workloads (**Scn4** and **Scn5**). Although the target systems of this thesis contain multipath e2e flows, only linear e2e flows are considered in these experiments, so that the results are not affected by other effects related to multipath e2e flows. This will provide a deep insight into the behavior of the algorithm.

In **Scn1**, four e2e flows (Γ_1-Γ_4) are allocated to four processors. The periods of their activation events, which represent a wide range of values, are as follows: $T_1 = 40$ ms, $T_2 = 250$ ms, $T_3 = 750$ ms and $T_4 = 1000$ ms. It is assumed that all their deadlines are equal to their periods, although the response time analysis technique would support arbitrary ones too. Therefore, Γ_1 is the most restrictive e2e flow and Γ_4 the least restrictive one. Table 6.1 shows, for each step of each e2e flow, its worst-case execution time. Best-case execution times are assumed to be the same as the worst-case ones.

In **Scn2** steps from Γ_1 are removed from CPU1 and in **Scn3** steps from Γ_2 are also relocated, thus obtaining an unbalanced partition allocation. Due to the MAF selection strategy described in Section 6.3.2, the solution space becomes independent in the context of each processor, depending on which is the most deadline-restrictive

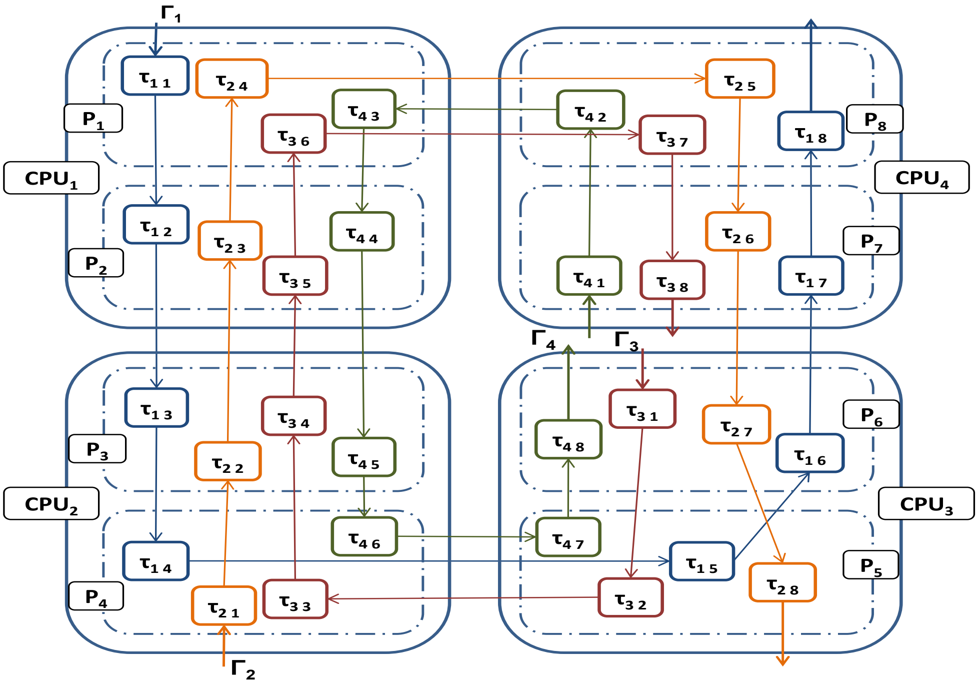


Fig. 6.7: Baseline synthetic scenario **Scn1**: 2 partitions hosted in each of the 4 processors

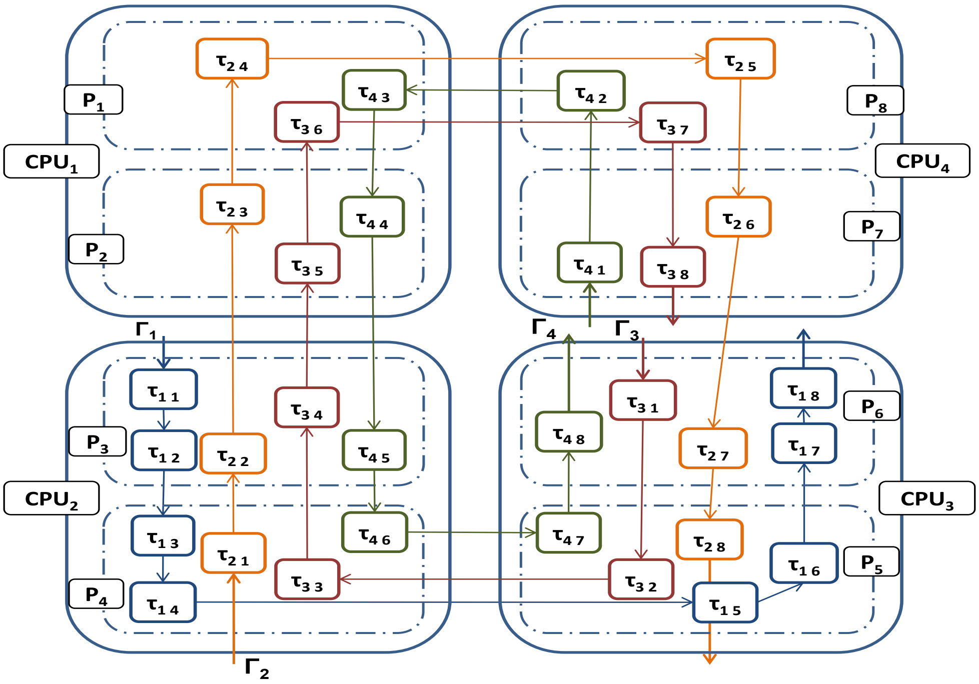


Fig. 6.8: Scn2: Γ_1 has been removed from CPU_1 and its steps relocated to partitions in the other processors

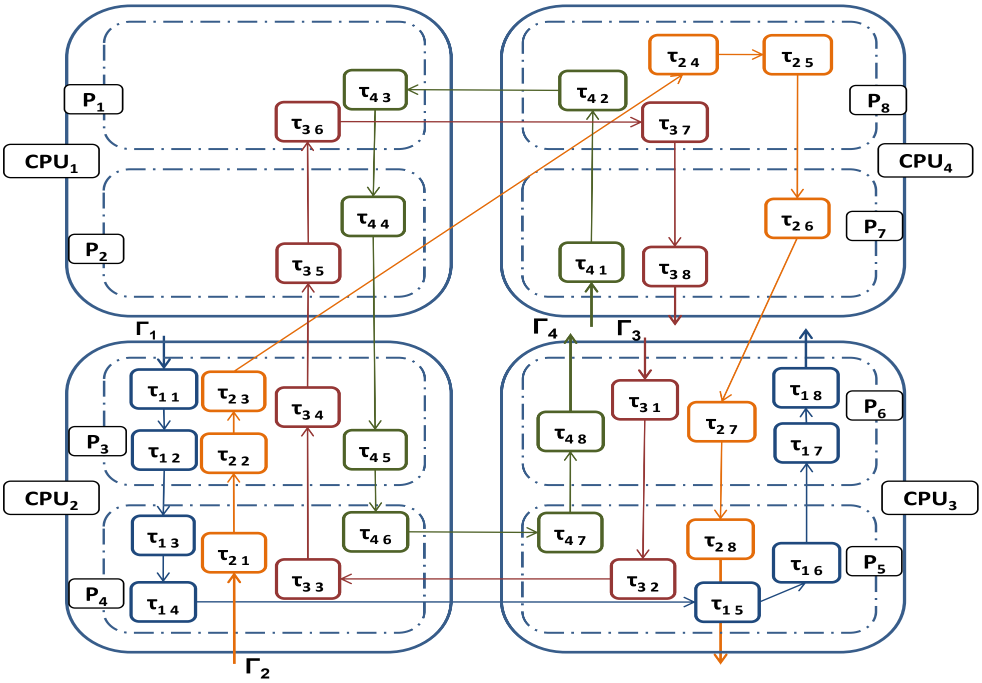


Fig. 6.9: Scn3: Γ_2 has also been removed from CPU_1 and its steps relocated to partitions in the other processors

Scn1					
e2e	Step	C_{ij}	e2e	Step	C_{ij}
Γ_1	τ_{11}	1	Γ_2	τ_{21}	1
	τ_{12}	0.5		τ_{22}	2
	τ_{13}	0.5		τ_{23}	3
	τ_{14}	1		τ_{24}	2
	τ_{15}	1		τ_{25}	1
	τ_{16}	2		τ_{26}	4
	τ_{17}	2		τ_{27}	2
	τ_{18}	1		τ_{28}	1
Γ_3	τ_{31}	15	Γ_4	τ_{41}	20
	τ_{32}	12		τ_{42}	40
	τ_{33}	20		τ_{43}	6
	τ_{34}	40		τ_{44}	7.5
	τ_{35}	30		τ_{45}	30
	τ_{36}	16		τ_{46}	37.5
	τ_{37}	18		τ_{47}	6.5
	τ_{38}	22		τ_{48}	11

Tab. 6.1: Worst-case execution times for Scn1

e2e flow they host. In **Scn4** and **Scn5** a scaling factor of 2 is applied to the worst-case execution times of the steps of the most restrictive and least restrictive e2e flows, Γ_1 and Γ_4 respectively, both considering the architecture of the baseline scenario. Since the algorithm considers the most restrictive flow's deadline to calculate the MAF in each processor, the effect of increasing the workload of Γ_1 has to be analyzed. Similarly, the effect of increasing the load in the least restrictive e2e flow is also important, as it has the least influence in the MAF calculation.

As the utilization of the partitions is in the range of 6% and 17%, the following available utilization for the partitions will be evaluated: 20%, 30%, 40% and 50% (note that 50% is the maximum available utilization possible as there are two partitions per processor). In these experiments, context switch overhead is assumed to be 1 μs.

6.3.2 WinAs algorithm characterization

After designing the experiments that will be carried out in this section, the WinAs algorithm will be applied on them in order to assess its performance in the different scenarios previously described.

The worst-case response times of each e2e flow, as a function of the number of iterations that WinAs takes, have been plotted in figures 6.10 to 6.14. In each figure, the e2e flows have been separated into different boxes, and each coloured plot represents a different available utilization assigned to each partition. The horizontal red line shows the deadline requirement of each e2e flow, and vertical axes have been represented in logarithmic scale for the sake of clarity. For each scenario, the first schedulable results, i.e. the MAF value for each processor, found by WinAs has been compiled in Table 6.2, along with the worst-case response times of the last step of each e2e flow.

Qualitative results are consistent with the ones obtained in the previous experiments, as for all e2e flows, increasing the number of windows is beneficial at the beginning of the search. However, and as was expected, it becomes detrimental (response times increase) earlier for the least restrictive e2e flows. Therefore, the number of window trade-offs must be carefully chosen in order to obtain a schedulable solution. In fact, notice that in Scn1, for 20% and 30% available utilizations, the algorithm fails to obtain a schedulable solution, as response-time curves of Γ_4 are always above the deadline plot. In contrast, when the available utilizations are 40% and 50%, the algorithm does find schedulable solutions.

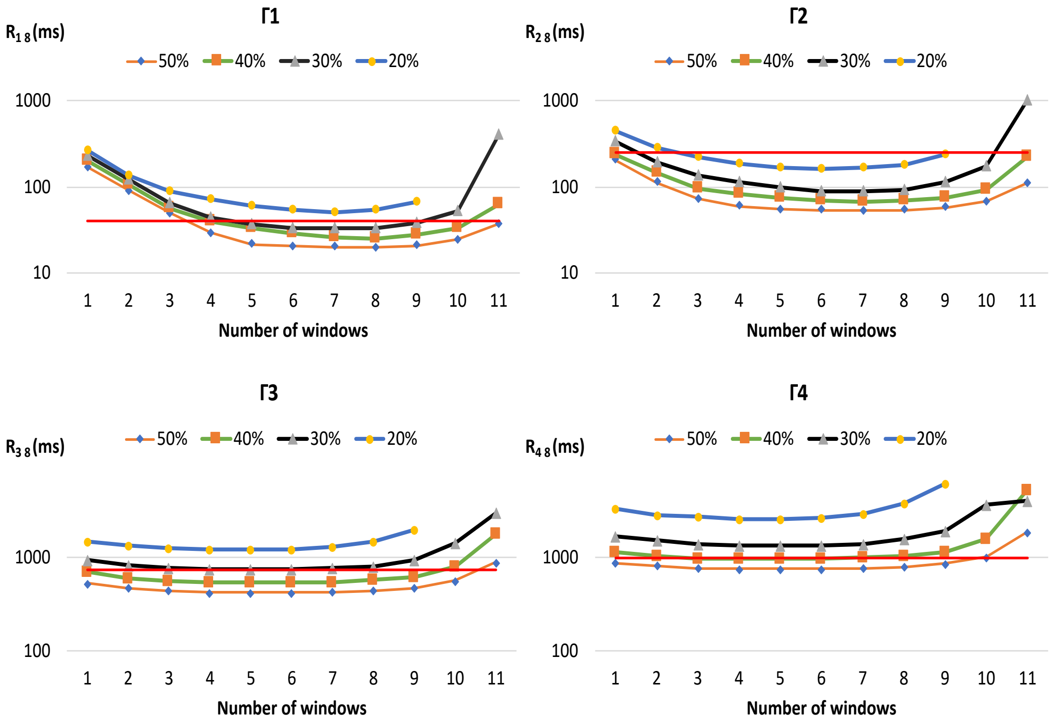


Fig. 6.10: e2e flows' worst-case response times for different numbers of windows (Scn1)

Regarding Scn2 and Scn3, the proposed algorithm takes more iterations to finish the search. During the first 10 iterations, the number of windows is increased in all processors, while after that, they are increased only in a subset of processors. This is because, in processors where the most restrictive e2e flows are hosted, the limit of the number of windows is reached before it is reached in those where they are not present. However, as shown in figures 6.11 and 6.12, response times always increase after the 10th iteration, when the number of windows is only increased in a subset of processors. Finally, as shown in Figures 6.13 and 6.14, which correspond to Scn4 and Scn5 respectively, although there are not significant qualitative differences in the behavior of the algorithm when the load of a certain e2e flow is increased, it fails in obtaining schedulable solutions for any available utilization at Scn5, and only when the available utilization is 50% there is a schedulable solution in Scn4. Generally, it can be stated that WinAs's behavior remains qualitatively consistent in all the scenarios described in this section.

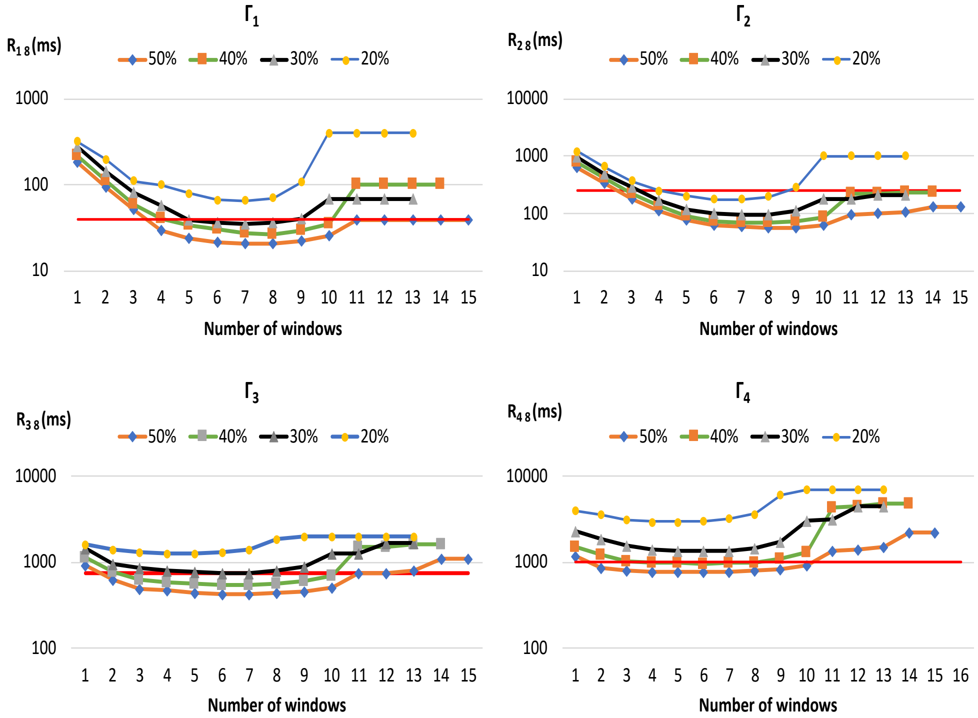


Fig. 6.11: e2e flows' worst-case response times for different numbers of windows (Scn2)

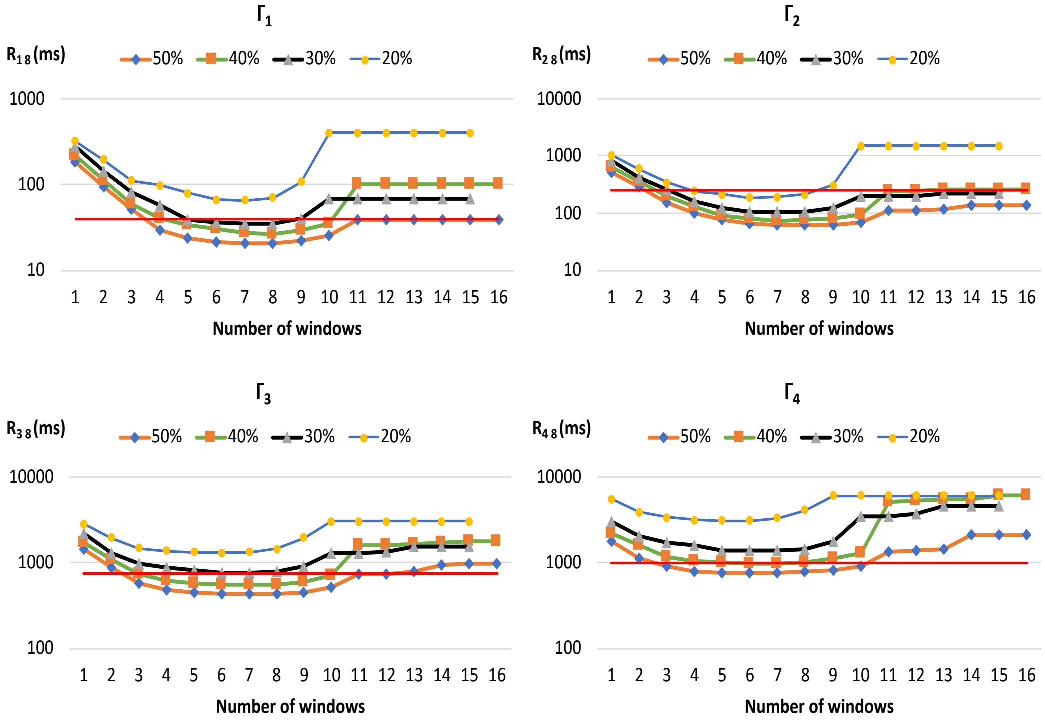


Fig. 6.12: e2e flows' worst-case response times for different numbers of windows (Scn3)

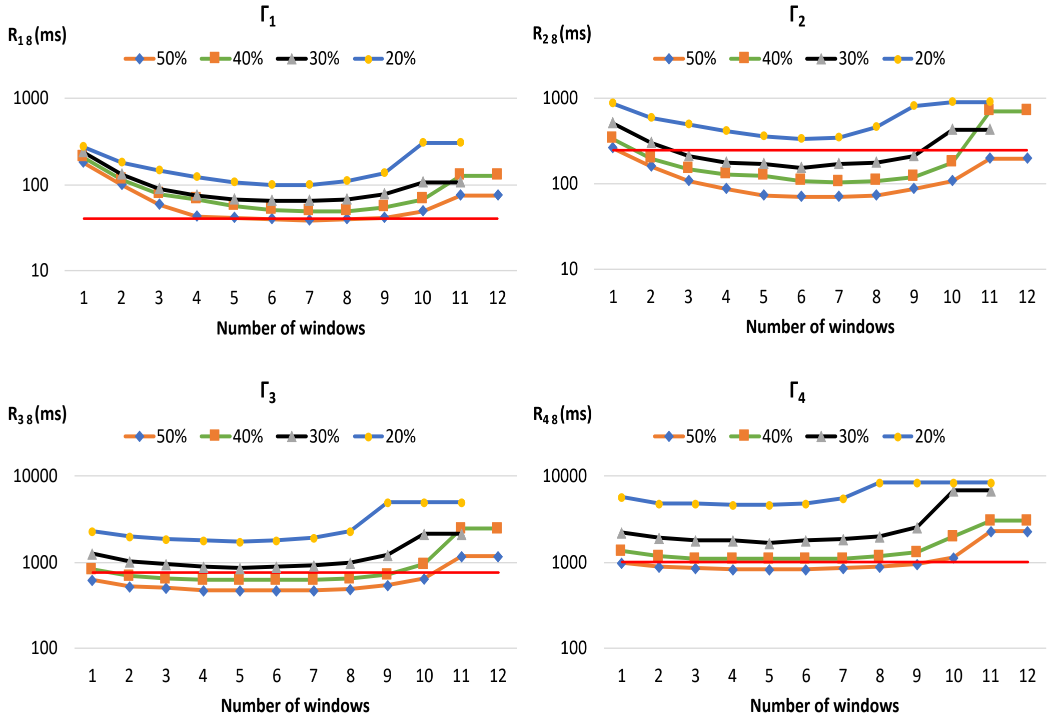


Fig. 6.13: e2e flows' worst-case response times for different numbers of windows (Scn4)

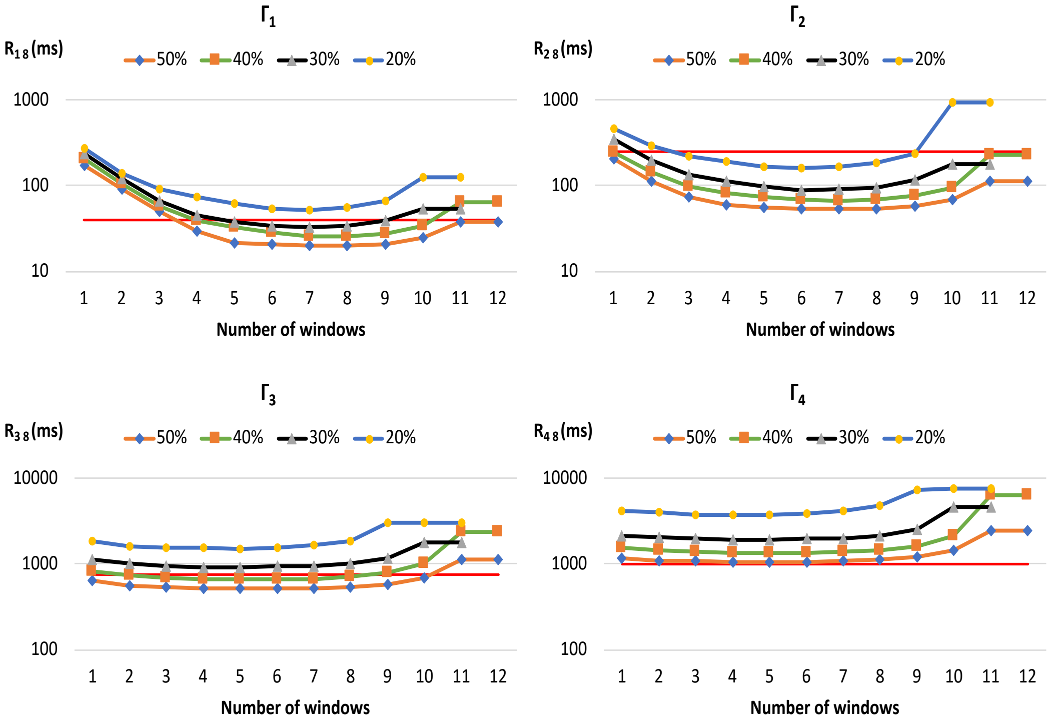


Fig. 6.14: e2e flows' worst-case response times for different numbers of windows (Scn5)

AU_{Px}	MAF (ms)				R_{ij} (ms)			
	CPU_1	CPU_2	CPU_3	CPU_4	$R_{1\ 8}$	$R_{2\ 8}$	$R_{3\ 8}$	$R_{4\ 8}$
Scn1								
40%	5	5	5	5	39.1	82.2	545.1	969.4
50%	5	5	5	5	29.08	60.1	423.8	757.03
Scn2								
40%	31.25	5	5	31.25	39.6	134.1	583.2	995.5
50%	31.25	5	5	31.25	29.58	110.1	467.1	773.4
Scn3								
40%	23.43	1.25	1.25	7.81	28.76	78.73	551.9	987.35
50%	93.75	5	5	31.25	29.58	102.01	485.29	795.07
Scn4								
50%	1.25	1.25	1.25	1.25	39.6	70.83	469.4	835.58

Tab. 6.2: Schedulable solutions found by WinAs: MAF and worst-case response times

6.3.3 Evaluating HOPWA algorithm

Once WinAs's performance has been fully characterized, HOPWA will be evaluated. To do so, the previously described test scenarios (Scn1 to Scn5) will be used. Since there are two partitions per processor, the main goal of this experiment is to assess the effect of the processing time initially allocated to each partition, i.e. the effect of assigning different AU_{Pi}^{init} values for each partition in each of the scenarios.

Configuration	AU_{P1}^{init}	AU_{P2}^{init}	AU_{P3}^{init}	AU_{P4}^{init}	AU_{P5}^{init}	AU_{P6}^{init}	AU_{P7}^{init}	AU_{P8}^{init}
$PC1$	80	20	80	20	80	20	80	20
$PC2$	50	50	50	50	50	50	50	50
$PC3$	20	20	20	20	20	20	20	20
$PC4$	30	10	30	10	30	10	30	10

Tab. 6.3: Initial available utilization (in %) for each partition in each partition configuration

The AU_{Px}^{init} values proposed for the evaluation are shown in Table 6.3. Four different partition configurations (PC1 to PC4) are considered, each of them being a paradigmatic partitioning scheme: PC1 represents an unbalanced distribution of processing time between the two partitions per processor, while in PC2 a fair distribution is performed; in PC3, a balanced distribution is also done, although some spare utilization is kept, and similarly in PC4 an unbalanced distribution is done while keeping some spare utilization as well.

As said before, HOPWA sets the available utilization for each partition as well as the MAF for each processor. These values, obtained after applying the algorithm to each scenario, are shown in Tables 6.4 to 6.8.

Scn1		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	Sched?
$PC1$	MAF (ms)	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25	×
	AU (%)	80	20	80	20	80	20	80	20	
$PC2$	MAF (ms)	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	✓
	AU (%)	33.5	33.9	35.1	34.7	33.1	34.5	35.5	34.7	
$PC3$	MAF (ms)	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	✓
	AU (%)	33.1	33.1	33.1	33.1	33.1	33.1	33.1	33.1	
$PC4$	MAF (ms)	5	5	5	5	5	5	5	5	✓
	AU (%)	48.7	28.7	48.7	28.7	48.7	28.7	48.7	28.7	

Tab. 6.4: MAF and AU values obtained by HOPWA, for each partition configuration in Scn1

Scn2		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	Sched?
$PC1$	MAF (ms)	31.25	31.25	5	5	5	5	31.25	31.25	×
	AU (%)	80	20	80	20	80	20	80	20	
$PC2$	MAF (ms)	15.6	15.6	2.5	2.5	2.5	2.5	15.6	15.6	✓
	AU (%)	32.5	33.4	36.08	35.2	35.0	35.5	33.7	33.8	
$PC3$	MAF (ms)	15.6	15.6	2.5	2.5	2.5	2.5	15.6	15.6	✓
	AU (%)	34.06	34.06	34.06	34.06	34.06	34.06	34.06	34.06	
$PC4$	MAF (ms)	31.25	31.25	5	5	5	5	31.25	31.25	✓
	AU (%)	50.6	30.6	50.6	30.6	50.6	30.6	50.6	30.6	

Tab. 6.5: MAF and AU values obtained by HOPWA, for each partition configuration in Scn2

Since all e2e flows cross all processors and partitions in Scn1, the calculated MAFs are the same for all configurations, as can be seen in Table 6.4. HOPWA cannot find any schedulable solution for PC1 due to the unbalanced available utilizations provided as input, while in PC2 the 50% of available utilization provided to each partition is enough to schedule the system, and the algorithm optimizes the solution by reducing the available utilization of each partition to the range of 33.1% and 35.5%. For PC3, where all partitions had an initial available utilization of 20%, HOPWA distributes the processors' spare utilization uniformly in all partitions,

Scn3		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	Sched?
$PC1$	MAF (ms)	23.4	23.4	1.25	1.25	1.25	1.25	7.81	7.81	×
	AU (%)	80	20	80	20	80	20	80	20	
$PC2$	MAF (ms)	11.7	11.7	0.62	0.62	0.62	0.62	3.9	3.9	✓
	AU (%)	37.8	38.3	40.7	39.8	39.7	40.1	38.8	39	
$PC3$	MAF (ms)	93.75	93.75	5	5	5	5	31.25	31.25	✓
	AU (%)	40.6	40.6	40.6	40.6	40.6	40.6	40.6	40.6	
$PC4$	MAF (ms)	23.4	23.4	1.25	1.25	1.25	1.25	7.81	7.81	✓
	AU (%)	50.6	30.6	50.6	30.6	50.6	30.6	50.6	30.6	

Tab. 6.6: MAF and AU values obtained by HOPWA, for each partition configuration in Scn3

Scn4		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	Sched?
$PC1$	MAF (ms)	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25	×
	AU (%)	80	20	80	20	80	20	80	20	
$PC2$	MAF (ms)	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	✓
	AU (%)	47.3	47.3	47.5	47.5	47.2	47.6	47.8	47.5	
$PC3$	MAF (ms)	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	✓
	AU (%)	47.6	47.6	47.6	47.6	47.6	47.6	47.6	47.6	
$PC4$	MAF (ms)	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	×
	AU (%)	60	40	60	40	60	4	60	40	

Tab. 6.7: MAF and AU values obtained by HOPWA, for each partition configuration in Scn4

Scn5		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	Sched?
$PC1$	MAF (ms)	5	5	5	5	5	5	5	5	×
	AU (%)	80	20	80	20	80	20	80	20	
$PC2$	MAF (ms)	5	5	5	5	5	5	5	5	✓
	AU (%)	48.9	49.0	49.1	49.1	48.9	49.0	49.1	49.1	
$PC3$	MAF (ms)	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	✓
	AU (%)	48.9	48.9	48.9	48.9	48.9	48.9	48.9	48.9	
$PC4$	MAF (ms)	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	×
	AU (%)	60	40	60	40	60	40	60	40	

Tab. 6.8: MAF and AU values obtained by HOPWA, for each partition configuration in Scn5

according to the methodology described in Section 6.3. After finding a schedulable solution, it is optimized by minimizing the available utilizations up to 33.1%. Finally, HOPWA finds a schedulable solution for PC4 despite the initial available utilizations being unbalanced, since there was some spare utilization to be distributed among all partitions that eventually allowed schedulability to be achieved. The differences between the solutions obtained for PC3 and PC4 are worth mentioning. In the former, the MAF obtained is 2.5 ms while in the latter it is 5 ms, which translates in half of context switch overheads. However, the schedulable solution for PC4 requires 48.7% available utilization for each partition while in PC3 the system can be scheduled with 15.6% less. The trade-off between the available utilization assigned to each partition and the context switch overheads should be carefully analyzed during the design of the system.

Tables 6.5 and 6.6, which are related to Scn2 and Scn3 respectively, correspond to those scenarios where some e2e flows do not traverse certain processors. For example, in Scn1 the MAFs obtained were 2.5 ms for PC2 and PC3, and 5 ms for PC4. However, in Scn2, where the most restrictive e2e flow is not present in a subset of processors, it is possible to keep the MAF at 15.6 ms in PC2 and PC3, and in 31.25 ms in PC4, minimizing context switch overheads significantly. The same effect can be observed in Scn3, where for instance in PC3 the MAF obtained is 93.75 ms in the processor where the two most restrictive e2e flows have been removed. This confirms that the proposed algorithm finds solutions that optimize the context switch overheads in those processors where it is possible to do so. Regarding the available utilizations, there are no significant variations in the solutions from Scn1 and Scn2-Scn3, and the reason why the values of MAF and AU from Scn2 and Scn3 are slighlty higher is due to the partitions' utilizations, which are slightly higher in the latter scenarios due to the relocation of some tasks.

In Scn4 and Scn5, whose corresponding results have been collected in Tables 6.7 and 6.8 respectively, partition configuration PC1 remains unschedulable as it was in the previous scenarios, and PC4 is also unschedulable in these scenarios. In Scn4, where the execution times of the most restrictive e2e flows were increased, the MAF values obtained for PC2 and PC3 are 0.6 ms in all processors, which are notably lower than the ones obtained in Scn1 (2.5 ms), thus implying higher context switch overheads. However, in Scn5, where the execution times of the least restrictive e2e flow were increased, the calculated MAFs are 5 ms for PC2 and 2.5 ms for PC3. It can be concluded that the most restrictive e2e flow influences the schedulability of the system to a greater extent than the least restrictive one.

Together with the results of applying the proposed algorithm under several scenarios and configurations, a sensitivity analysis is also performed as part of this assessment, in order to compare and characterize the solutions obtained. The following parameters defined in Section 3.4 will be used: System Slack Factor (SSF), Partition Slack Factor (PSF) and e2e Flow Slack Factor (FSF). These sensitivity analysis results are shown in a table for each scenario: 6.9 to 6.13 for Scn1 to Scn5 respectively.

Regarding Scn1, all SF values corresponding to configuration PC1 are lower than 1 as the system is not schedulable. As shown in Table 6.9, when the system is schedulable (PC2, PC3 and PC4), partition slack factors are all very small, since most of the values are near 1, which is illustrative of the tightness of the solutions that the algorithm provides. This is also confirmed by the very low system slack factors obtained. In general, the e2e flow slack factors are higher for the most restrictive e2e flows. This is coherent, as the algorithm tries to schedule the most restrictive e2e flows first by selecting an appropriate MAF, and therefore the least restrictive ones suffer an increase in their response times, as shown in the study of Section 6.2. Comparing the results for PC3 and PC4, it can be stated that the solution obtained by HOPWA for PC4, i.e. higher MAF and higher available utilization, is better than the one obtained for PC3, because all SF values are higher in PC4. Although it was stated that the most restrictive flow determines the schedulability of the system to a greater extent, the sensitivity analysis shows that the schedulable solutions found by HOPWA scale better for Scn4 than for Scn5, since the calculated SF values are always slightly higher. The trade-offs between the context switch overheads and the partitions' spare utilization on the one hand, and the scalability of the solutions obtained by HOPWA on the other, are aspects that need to be considered during the system design.

Regarding Scn4 and Scn5, only two partition configurations, PC2 and PC3, enable schedulable solutions to be reached as shown in Tables 6.12 and 6.13. In Scn4, the MAF obtained for scheduling the system is notably smaller (0.6 ms in all processors) than the ones in Scn1 (6.9 ms in PC1 and 2.5 ms in PC2). However, there is a positive effect in the slack factors of those e2e flows which have not been moved, so it can be stated that decoupling the restrictive flow from one processor may be beneficial as the effect context switch overhead is reduced.

SF		PC1	PC2	PC3	PC4
SSF		0.88	1.052	1.007	1.06
PSF	P1	0	1.68	1.27	2.42
	P2	0	1.48	1.13	1.54
	P3	0	1.23	1.02	1.36
	P4	0.28	1.28	1.02	1.2
	P5	0	1.81	1.14	2.4
	P6	0.2	1.44	1.01	1.39
	P7	0	1.3	1.03	1.2
	P8	0.55	1.28	1.03	1.2
FSF	Γ_1	0.004	1.04	1.07	1.2
	Γ_2	0	1.06	1.1	2.2
	Γ_3	0.58	1.06	1.07	1.17
	Γ_4	0.72	1.005	1.09	1.11

Tab. 6.9: Slack factors (system, partitions and e2e flows) for Scn1

SF		PC1	PC2	PC3	PC4
SSF		0.84	1.028	1.005	1.014
PSF	P1	0	1.2	1.32	3.29
	P2	0	1.03	1.01	1.75
	P3	0	1.03	1.01	1.46
	P4	0	1.04	1.02	1.29
	P5	0	1.02	1.12	1.26
	P6	0.34	1.03	1.03	1.01
	P7	0	1.04	1.04	1.79
	P8	0	1.01	1.06	1.43
FSF	Γ_1	0.007	1.17	1.04	1.01
	Γ_2	0	1.7	1.04	3.01
	Γ_3	0	1.07	1.01	1.2
	Γ_4	0	1.06	1.01	1.1

Tab. 6.10: Slack factors (system, partitions and e2e flows) for Scn2

SF		PC1	PC2	PC3	PC4
SSF		0.75	1.004	1.015	1.014
PSF	P1	0	1.04	1.73	2.15
	P2	0	1.07	1.01	1.53
	P3	0	1.03	1.1	1.13
	P4	0	1.04	1.13	1.08
	P5	0	1.05	1.2	1.26
	P6	0	1.07	1.16	1.01
	P7	0	1.07	1.11	1.18
	P8	0	1.05	1.08	1.08
FSF	Γ_1	0	1.11	1.13	1.13
	Γ_2	0	1.2	1.74	1.75
	Γ_3	0.37	1.06	1.02	1.08
	Γ_4	0.42	1.01	1.05	1.06

Tab. 6.11: Slack factors (system, partitions and e2e flows) for Scn3

SF		PC1	PC2	PC3	PC4
SSF		0.64	1.002	1.007	0.99
PSF	P1	0	1.001	1.04	0.99
	P2	0	1.04	1.1	0.96
	P3	0	1.004	1.1	0.84
	P4	0	1.005	1.04	0.96
	P5	0	1.02	1.04	0.96
	P6	0	1.007	1.02	0.96
	P7	0	1.01	1.02	0.96
	P8	0	1.004	1.04	0.96
FSF	Γ_1	0	1.002	1.007	0.99
	Γ_2	0	5.1	5.1	0
	Γ_3	0	1.45	1.43	0
	Γ_4	0	1.63	1.62	0

Tab. 6.12: Slack factors (system, partitions and e2e flows) for Scn4

SF		PC1	PC2	PC3	PC4
SSF		0.53	1.002	1.002	0.95
PSF	P1	0	1.013	1.05	0.99
	P2	0	1.05	1.02	0
	P3	0	1.01	1.005	0.66
	P4	0	1.01	1.008	0.78
	P5	0	1.06	1.05	0
	P6	0	1.03	1.02	0.54
	P7	0	1.03	1.006	0.55
	P8	0	1.01	1.005	0.8
FSF	Γ_1	0	1.03	1.02	0.37
	Γ_2	0	1.11	1.05	0
	Γ_3	0	1.01	1.007	0.8
	Γ_4	0	1.003	1.002	0.93

Tab. 6.13: Slack factors (system, partitions and e2e flows) for Scn5

6.3.4 Scheduling evaluation of the industrial use-case

The partition scheduling problem addressed in this work is motivated by a safety-critical railway application, whose architecture has been described in Chapter 1 and whose most relevant time-related features have been modeled in Chapter 3 and depicted in Figure 3.4. Now, the proposed HOPWA algorithm is applied to this scenario.

Although the safety standards oblige completion of the execution of functionalities within 1 s, in this section a wider range of deadlines will be evaluated, in order to determine the behavior of the algorithm under more constrained timing requirements. Therefore, four different values of the e2e deadlines (5, 50, 500 and 1000 ms) will be explored. Similarly to Section 6.2.4, the objective of this evaluation is to characterize the behavior of HOPWA under different AU_{Px}^{init} ranges, which are its input parameters. As the application represents a very low load, the initial available utilizations to be tested will be 2% for the processing partitions and 1% for the communications partitions. In addition, a higher order of magnitude (20% and 10%) of available utilizations will also be evaluated. Table 6.14 shows the two configurations just described, and the results of applying HOPWA to the industrial use-case are shown in Tables 6.15 and 6.16.

Partition / Config.	AU_{P1}^{init}	AU_{P2}^{init}	AU_{P3}^{init}	AU_{P4}^{init}
Config.1	20	10	20	10
Config.2	2	1	2	1

Tab. 6.14: Initial available utilization (in %) for each partition in each Configuration (Config.)

Table 6.15 shows the outcome of HOPWA for each deadline value (in ms) and partition configuration (Config.1 and Config. 2). Thus, as in previous experiments, the MAF calculated at each processor and the AU for each partition are shown. Furthermore, as the whole application is modeled as a single e2e flow, the MAFs in all processors are the same. As the deadline becomes tighter, HOPWA provides more available utilization to partitions and decreases the period of the MAF in order to achieve schedulability.

It can be seen that the algorithm is able to optimize the solution obtained by minimizing the available utilizations provided as inputs. In Table 6.16, the worst-case response times of the tasks that represent the output of the three functionalities have been compiled: $\tau_{1\ 11}$ and $\tau_{1\ 13}$ for the EB functionality, $\tau_{1\ 23}$ and $\tau_{1\ 25}$ for the RBC-CS functionality and $\tau_{1\ 35}$ and $\tau_{1\ 37}$ for the PV-DMI functionality. The algorithm

Deadline (ms)	Configuration	Params.	P1	P2	P3	P4
1000	Config.1	MAF (ms)	125	125	125	125
		AU (%)	5	2.5	5	2.5
	Config.2	MAF (ms)	125	125	125	125
		AU (%)	0.5	0.25	0.5	0.25
500	Config.1	MAF (ms)	62.5	62.5	62.5	62.5
		AU (%)	5	2.5	5	2.5
	Config.2	MAF (ms)	62.5	62.5	62.5	62.5
		AU (%)	0.5	0.25	0.5	0.25
50	Config.1	MAF (ms)	62.5	62.5	62.5	62.5
		AU (%)	5	2.5	5	2.5
	Config.2	MAF (ms)	3.12	3.12	3.12	3.12
		AU (%)	0.76	0.37	0.76	0.37
5	Config.1	MAF (ms)	0.3	0.3	0.3	0.3
		AU (%)	7.5	3.7	7.5	3.7
	Config.2	MAF (ms)	0.3	0.3	0.3	0.3
		AU (%)	14	13	14	13

Tab. 6.15: MAF and AU values obtained by HOPWA for the railway use-case

D_{ij}	Config.	$\tau_{1\ 11}$	$\tau_{1\ 13}$	$\tau_{1\ 23}$	$\tau_{1\ 23}$	$\tau_{1\ 35}$	$\tau_{1\ 37}$
1000	Config.1	718.842	962.59	718.846	962.592	718.86	962.595
	Config.2	746	996.3	746.9	996.34	746.95	996.35
500	Config.1	359.4	481.3	359.5	481.4	359.51	481.42
	Config.2	373	498.2	373.5	498.3	373.6	498.4
50	Config.1	36.12	48.3	36.127	48.31	36.123	48.32
	Config.2	40.5	46.8	40.54	46.81	40.56	46.82
5	Config.1	4.284	4.283	4.002	4.285	4.314	4.317
	Config.2	2.348	2.886	2.358	2.896	2.378	2.916

Tab. 6.16: Worst-case response times (in ms) of the railway application under the different configurations and deadlines tested

always finds a partition scheduling that enables the fulfillment of the application's deadline.

6.4 Conclusion

In this chapter, the scheduling of time partitions has been addressed. First, a study of the effect of the size and number of partition windows has been performed. The knowledge coming from this study has been leveraged to develop an algorithm, called WinAs, to search for schedulable solutions by adjusting the MAF with one window per partition for a fixed available utilization, and then it has been integrated within the Heuristic Optimized Partition Window Assignment (HOPWA) algorithm. HOPWA receives an initial available utilization for each partition, executes WinAs internally for every attempt over a fixed available utilization, and also tunes the available utilization of each partition. The proposed algorithm has been applied to a representative set of synthetic experiments and also to the industrial use-case that motivates this thesis. It is shown to be capable of finding schedulable solutions in many different scenarios, including applications composed of a wide range of deadline requirements.

As highlighted in the chapter, there are quite a few research paths to consider. Exploring a non-uniform window assignment would be one of the first tasks to address, together with different strategies of MAF reduction inside WinAs, depending on specific application features. All the knowledge gathered during the study, in combination with the results from the heuristic approach, also provide a solid basis for developing more complex search and optimization algorithms, such as simulated annealing or genetic algorithms.

Step-to-processor allocation 7

Allocating steps to processors is, as shown in the literature review of Chapter 2, a complex problem that has been widely addressed. Although the allocation problem initially was beyond the scope of this thesis, the opportunity to explore some preliminary solutions in the field of general distributed/multicore real-time systems arose, motivated by the novel architectures and design paradigms found in, for example, the automotive or robotics domains.

The use of heterogeneous multicore processors is a clear example of the new trends in the design and implementation of safety-critical applications, as many theoretical developments in the context of real-time analysis and scheduling needed to be reinvented in order to face the new challenges that arose [FGB10][BBB15][Cap+20]. Moreover, the advent of these heterogeneous architectures, where differently featured processors are integrated within the same computing platforms, has demonstrated that traditional approaches are no longer efficient to achieve valid solutions in this context. In this chapter, it is shown that the step-to-processor allocation in real-time systems is a paradigmatic case of this problem, and a new method to address it is presented. Although this is only the first step, the preliminary experiments show promising results and highlight the good direction of the proposed approach.

7.1 Background

The allocation of steps to processors in distributed real-time systems has been extensively addressed, making use of algorithmic approaches of different complexity. Allocating real-time applications into computing platforms composed of more than one processor can be understood as a bin packing problem [Har82]. This is an NP-hard optimization problem in which different sized items, in this case real-time tasks, are placed into a finite and known number of bins, in this case processors. Bins have a known capacity that must not be exceeded, in other words, their utilization must not exceed 100%.

In general, bin packing algorithms operate by sorting the steps, and in each stage of the algorithm, one of the steps in the sort is placed in a processor according to different criteria. The following three well-known bin packing algorithms, which consider the processor utilization as a reference parameter, will be explored in this chapter:

- Worst-Fit: In each stage, steps are allocated to the processor with the lowest utilization among all the processors that could host them.
- Best-Fit: Steps are allocated to the processor with the highest utilization that can host them.
- First-Fit: Steps are allocated to the first processor found that can host them.

The way of sorting steps has a strong impact on the result, as will be shown later, and it can be performed in different ways, such as sorting them according to their utilization or their relative positions in memory.

The core of many state-of-the-art allocation approaches mentioned in the literature review in Chapter 2 is based on these bin-packing algorithms, such as [Gar+15] or [Cas+18].

7.1.1 Multicore achitectures

Two kinds of scheduling policies are traditionally considered in multiprocessor/multicore systems [BBB15] [DB11]: in global scheduling, steps can be executed in any core, thus allowing migration from one to another core during runtime, while in partitioned scheduling, the step-to-core allocation is fixed and migrations are not allowed. In [Ber+20] a hybrid approach is considered, where processors are grouped into clusters and step migrations are only allowed in the context of such clusters. In this chapter, due to the target applications addressed, a partitioned scheduling policy is considered. Safety critical applications can take advantage of multicore devices in order to isolate a certain number of cores for real-time processing, as well as for isolating different criticality components. In any case, statically defined step-to-processor mapping is necessary for these applications. Initially, multicore architectures were composed of a set of cores with identical features, in terms of processing speed. These are known as homogeneous architectures. However, nowadays real-time systems need to make use of more sophisticated devices in order to perform computing-intensive duties [OCC18], such as the execution of computer vision inference algorithms, combined with different non-functional requirements, for instance power management control. Because of that, multiprocessor systems are nowadays integrating cores of different processing speeds and capacities, giving rise to heterogeneous architectures.

In this work, heterogeneous multiprocessor systems where each core may have a different processing speed are considered. With the advent of the artificial intelligence deployed in the automotive industry as a key enabler of autonomous driving, it is mandatory to make use of more sophisticated devices to support the high computational necessities. For instance, the NVIDIA Jetson TX2 board, which is at the base of the NVIDIA board series for autonomous driving and ADAS (Advanced Driving Assistance Systems), was presented in [Wur+19][Reh+21], where the authors proposed an initial modeling of its performance, as a paradigmatic example of the heterogeneous architectures described here. This device is composed of three clusters or islands, where differently featured processors are placed: a four-core island runs at 1.9 GHz, a two-core island runs at 2 GHz and the third island hosts a GPU for high computational necessities.

The response time analysis technique, as well as the scheduling techniques developed in the context of this thesis, can be directly applied to multiprocessor systems based on partitioned scheduling.

7.1.2 Allocating real-time applications in heterogeneous systems

The heterogeneity in processing speeds just described makes most of the allocation algorithms in literature unsuitable, since each step may have a different worst-case execution time depending on the processor it is allocated to. For example, the traditional Best-Fit algorithm is commonly used to minimize the number of processors used, while Worst-Fit aims to produce balanced allocations. However, since they are not aware of anything other than the processors' utilization, they may take decisions that severely undermine the worst-case response times, by placing the steps in processors where their execution times explode.

To tackle this, most approaches that address heterogeneous systems' allocation rely on complex algorithms that try to find optimized solutions, if any. In [Höt+19] the authors developed a genetic algorithm and different fitness functions, depending on the feature to be optimized: response times, end-to-end latencies or processor load balancing. In [Cas+19] a MILP formulation is proposed to solve the allocation problem in a heterogeneous architecture like the one just presented. More recently, several LP and ILP methods to allocate tasks to processors are presented in [Ber+21], where target platforms are categorized as unrelated multicore architectures. These methods, however, scale very poorly when the complexity of the systems increases.

7.2 Slack-Based Allocation (SBA) algorithm

The design of the Slack-Based Allocation (SBA) algorithm, proposed in Algorithm 10 for step-to-processor allocation in distributed real-time systems, is inspired by the previously discussed bin-packing algorithms. Its input is a distributed system based on preemptive FP scheduling and composed of steps that are not allocated to any processor. Its output is the step-to-processor allocation together with a priority assignment for each step.

As mentioned before, traditional bin-packing allocation algorithms have the utilization of processors as a reference to decide where to place the steps in each stage. Instead of that, with the aim of capturing the effect on the worst-case response times of the decisions that the allocation algorithm takes in each stage, the following slack factor parameters defined in Section 3.4 will be considered: System Slack Factor (SSF) and Processor Slack Factor. These parameters give an idea of the goodness (in terms of schedulability) of the decision of allocating a step to a certain processor, and they will be computed to determine where to place the steps in each stage of the algorithm.

In each stage of the algorithm, a $Candidate\_List$ is created, which gathers the set of processors where the step under assignment can be allocated. First, steps are sorted in a certain manner, and in each allocation stage one of them (the first one in the sorting) is allocated into a processor. As will be explained later, the proposed algorithm has to calculate the slacks at each allocation stage, which implicitly means that response-time analysis should be carried out. This kind of analysis has to be executed over different subsets of steps which must be compliant with the e2e flow model. Therefore, step-to-processor allocation must be performed in such a way that steps' precedence relations are preserved. To achieve this, steps will be sorted according to their priorities, which must be assigned in a non-decreasing order from the first step until the end. This particular priority assignment can be achieved by a subset of the algorithms developed in this thesis. As discussed in Chapter 5, some of the algorithms developed for priority assignment produce so-called global deadlines, which in combination with the proposed method to transform VDs into priorities, can guarantee that the resulting priority assignment is topological and non-decreasing. From all the algorithms developed in Chapter 5, PD_Local cannot be applied in this context since it produces local VDs and therefore non-decreasing priorities cannot be assured. NPD_Global and NPD_Local depend on the utilization of the partition/processor where the step is hosted, and so they are not applicable in this context where the allocation is being defined. Finally, it is not guaranteed that the priorities resulting from applying EQF will be non-decreasing, since the expression to calculate the VDs includes a proportionality factor, so it is discarded too. Therefore UD, ED, PD_Global and EQS are the four priority assignment algorithms that can be applied to sort the steps in the proposed allocation method.

Algorithm 10: Slack-based Allocation Algorithm

```
 1: Input:  Set of steps $\tau_{ij}$, set of Processors $CPU_y$
 2: Priority Assignment
 3: Sort all $\tau_{ij}$ according to $Prio_{ij}$
 4: for each $\tau_{ij}$ do
 5:   $MaxSSF = 0$ , Clear $Candidate\_List$
 6:   for each $CPU_y$ do
 7:     $CPU_y \leftarrow \tau_{ij}$
 8:     Calculate System Slack
 9:     if $MaxSSF <$ System_Slack then
10:       $MaxSSF$ = System_Slack
11:       Clear $Candidate\_List$ & $Candidate\_List \leftarrow CPU_y$
12:     else if System_Slack = $MaxSSF$ then
13:       $Candidate\_List \leftarrow CPU_y$
14:     end if
15:   end for
16:   if $CPU_y \in Candidate\_List$ > 1 then
17:     $MaxCPUSF = 0$
18:     for each $CPU_y \in Candidate\_List$ do
19:       $CPU_y \leftarrow \tau_{ij}$
20:       Calculate Processor_Slack at $CPU_y$
21:       if Processor_Slack > $MaxCPUSF$ then
22:         $CPU_y \leftarrow \tau_{ij}$
23:       end if
24:     end for
25:   else
26:     $CPU_y \in Candidate\_List \leftarrow \tau_{ij}$
27:   end if
28: end for
```

At each stage (Line 4), the algorithm allocates the step in all processors and calculates the resulting system slacks (Line 8). If there is a single processor where the maximum SSF (*MaxSSF*) is achieved, the step is allocated to that processor (Line 26), whereas if there is more than one, all of them are added to the $Candidate\_List$ and their CPUSFs are calculated (Line 20). The step will be allocated to the processor where the maximum CPUSF (*MaxCPUSF*) is achieved, and if there is still a tie at processor SF, the allocation is decided arbitrarily, subject to future optimization strategies to be evaluated.

Although the design of the algorithm seems rather simple, it is computationally intensive, as the slack calculation is based on iteratively applying the response time analysis, which may explode with the size of the target real-time application (the number of steps and the number of processors). However, the calculations of slack parameters are independent among themselves, and they can be performed in parallel. The authors in [RGH17] proposed a supercomputing framework that can be leveraged for launching system and processor slack calculations in parallel and gathering the results to speed-up the algorithm's performance. This will be proposed as future work.

7.3 Preliminary evaluation

As said before, the work contained in this chapter is only a first step in the application of the techniques developed in this thesis to the allocation problem. Even if it is not fully related to the industrial use-case addressed in this thesis, many of the techniques developed and information contained in this thesis have been successfully applied to this topical problem. In this section, a preliminary evaluation of the proposed allocation algorithm is carried out on three different test cases. In this first approach, homogeneous processors in terms of processing speed are considered.

The preliminary experiments are based on randomly generated synthetic e2e flows, generated following the principles in [Mel+15]. A special purpose generation tool has been developed and it has also been made publicly available[1]. It is based on the well-known DAG model [LA10][Ver+20], which can be directly transformed to the multipath flow model described in this thesis. The utilization of the generated e2e flows can be provided as input parameters to the tool, as well as the number of flows and the steps within them. Test cases consist of different sized e2e flows, in terms of the number of steps, which are allocated individually to a varying number of processors. A test case will be developed through the execution of the proposed algorithm to allocate a single e2e flow into a set of processors. A hundred different e2e flows are tested for incremental utilizations in a specific range with a fixed step. In all experiments, the SBA algorithm is compared against the Worst-Fit algorithm considering two variations, one where steps are sorted in decreasing utilization order (WF_D) and the other where steps are sorted in a topological arbitrary order (WF_Topo). If the opposite is not stated, the priority assignment algorithm used to sort the steps at the beginning of the algorithm will be PD_Global.

[1] https://github.com/mive93/DAG-scheduling

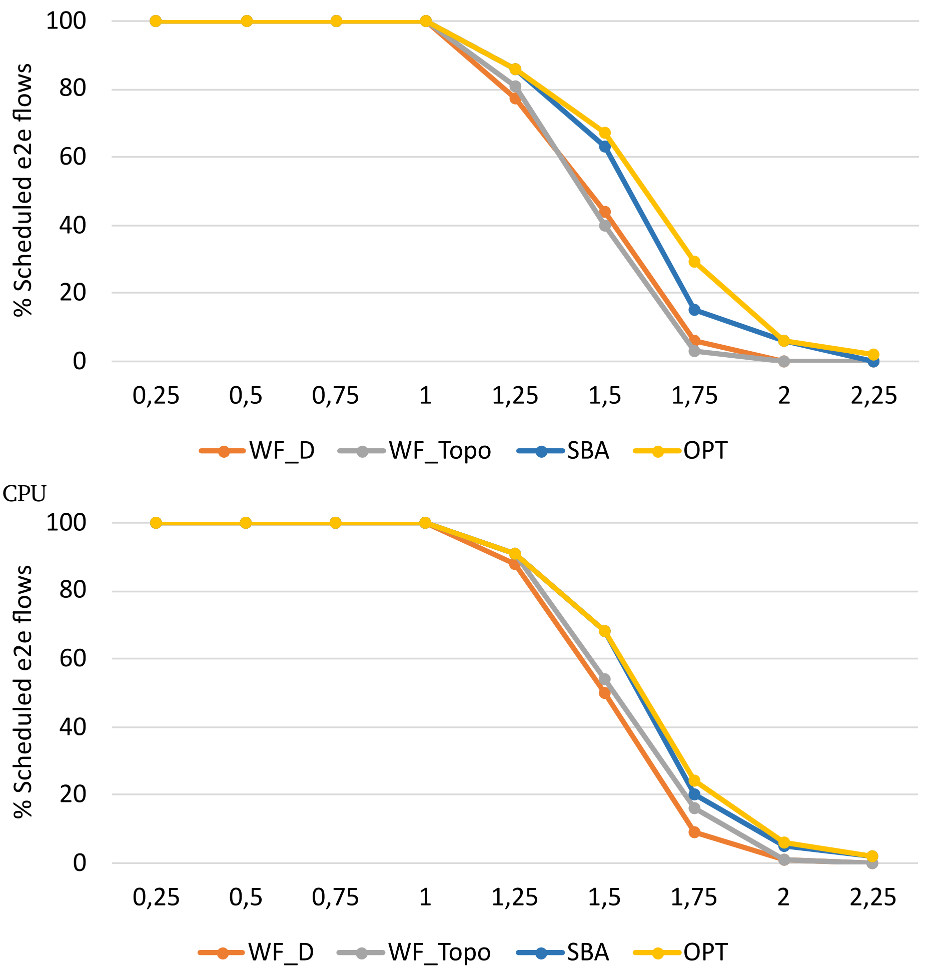
The first test case consists of small sized e2e flows, where the number of steps in each e2e flow is in the range [7,10]. The utilization of the e2e flows has been set from 0.25 to 2.5, with an increasing step of 0.25, and they will be allocated to 3 and 4 processors. Notice that the generated e2e flows' utilization may be higher than 1, as their load is will be distributed in several processors. The optimal solution, i.e the lowest worst-case response time among all the possible allocation solutions, will also be evaluated, in order to see how close the solutions obtained are to the optimal one. The optimal solution is obtained by checking all the possible solutions through a recursive backtrack method, where the unfeasible solutions, i.e. those whose processor utilization is over 1, are discarded before computing the worst-case response-time.

In the second test case, corresponding to medium-sized e2e flows, the input value in the generation tool is set so that the generated flows are composed of a number of steps in the range [10,25]. For this experiment, the e2e flows will be allocated to 3, 4 and 5 processors. The e2e flows' utilizations are from 0.33 to 3.33 with an increasing step of 0.33. Due to the larger size of this experiment, it was not possible to obtain the optimal solution through the previously described method, so the optimal solution's reference is no longer available for these results.

Finally, a large-sized e2e flow experiment is performed by generating flows with a number of steps in the range [50-100], with utilization values from 0.75 to 5 with an increasing step of 0.25. They will be allocated to 4, 5 and 6 processors. In this test case, another feature of interest will be evaluated. As stated before, steps are initially sorted according to their priorities, which are assigned following different methods. In order to assess the impact of the priority assignment on the initial sorting, and therefore on the allocation algorithm, in this experiment the schedulability results will be presented for different step orderings. The evaluated priority assignment algorithms will be ED (SBA_ED), PD_Global (SBA_PD) and UD (SBA_UD).

As said before, the SBA algorithm will be compared with two bin-packing algorithms, WF_D and WF_Topo, and also with the optimal solution in small-sized experiments. The results have been plotted in Figures 7.1, 7.2 and 7.3. They show, for each utilization value, the percentage of e2e flows that meet their deadlines after being allocated through the different methods. In all figures, each graph represents a different number of processors where the e2e flows have been allocated.

In Figure 7.1, the SBA algorithm (yellow plot) outperforms the bin packing algorithms and obtains near-optimal solutions for the small-sized test case. As shown in 7.2, the proposed algorithm still produces more schedulable results than the WF algorithms evaluated for all numbers of processors. Figure 7.3 shows that the SBA



(a) 3 CPU

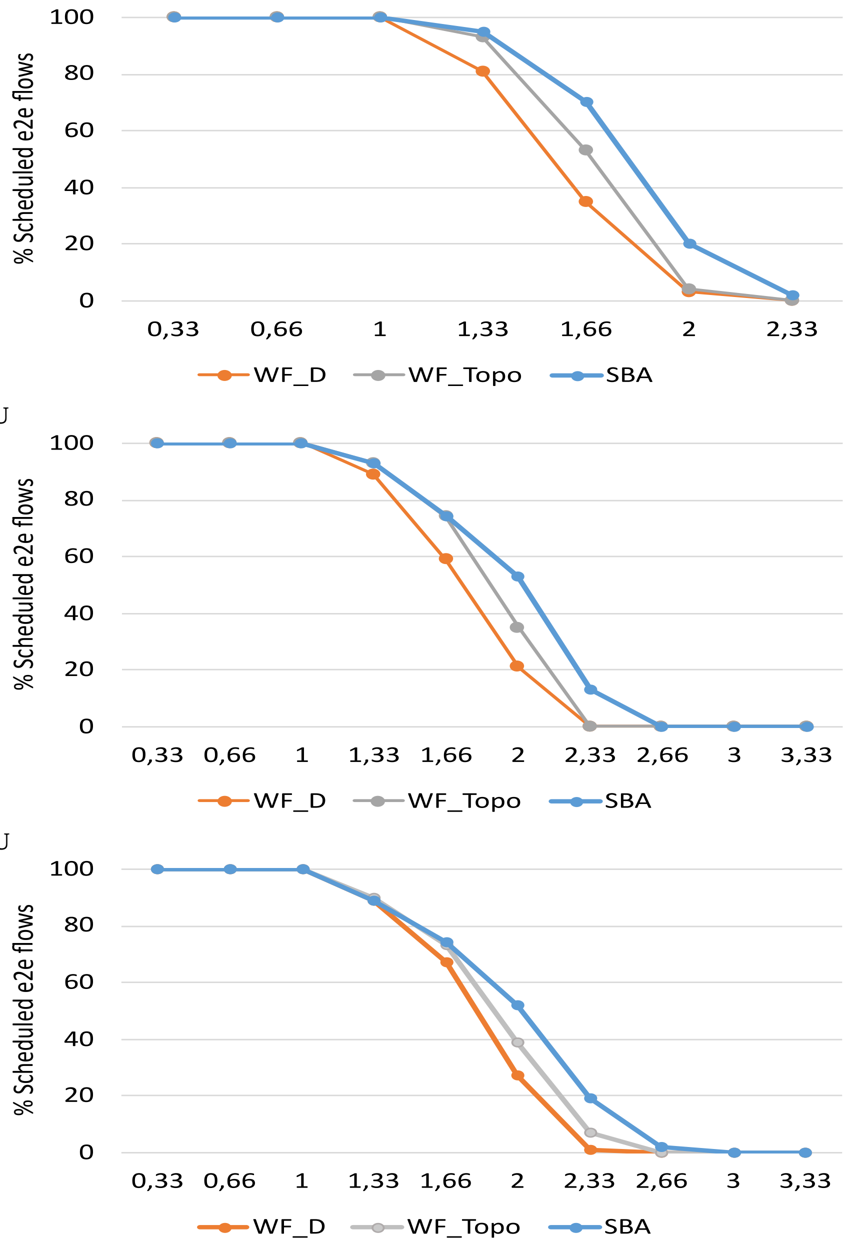
(b) 4 CPU

Fig. 7.1: Small-sized test

algorithm remains better than the bin packing algorithms for all the utilization values and number of processors evaluated. Moreover, it can be seen that the initial sorting of steps, performed via different priority assignment algorithms, has a paramount importance in the schedulability of the synthetic applications generated. For any number of processors, the initial step sorting produced by using the ED algorithm shows the best performance, so future research works should consider this as a reference.

7.4 Conclusions

Motivated by new hardware architectures, such as heterogeneous multicore systems that enable the execution of sophisticated applications, this work aims to take the first step towards new allocation algorithms. Preliminary results are promising, so the slack can be considered as an important reference parameter to tackle the allocation of real-time applications in such complex computing platforms. As pointed out in the chapter, future works will address the optimization of slack calculations by means of a supercomputer, as well as comprehensive experiments to fully characterize the proposed algorithm.

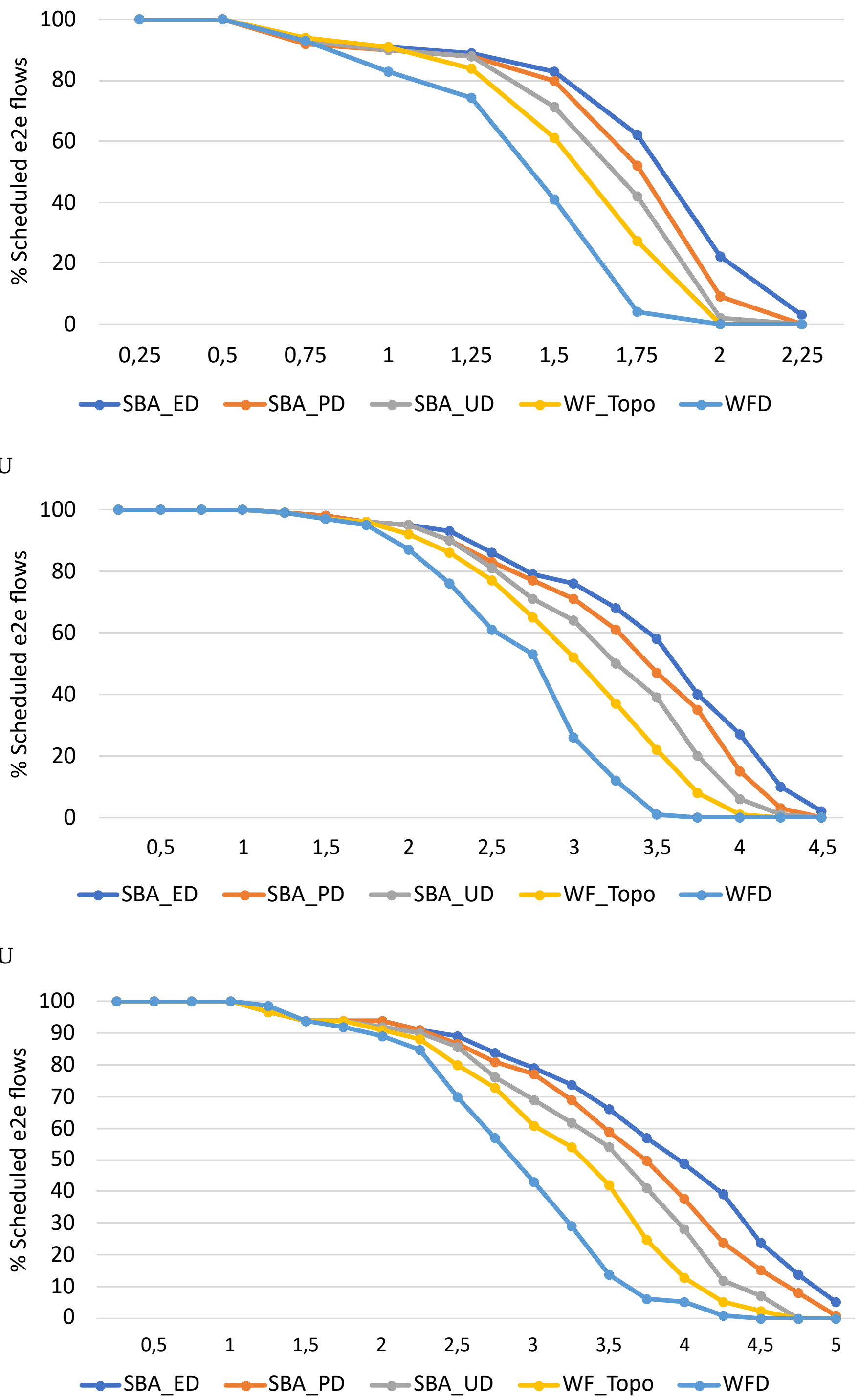


(a) 3 CPU

(b) 4 CPU

(c) 5 CPU

Fig. 7.2: Medium-sized test



(a) 4 CPU

(b) 5 CPU

(c) 6 CPU

Fig. 7.3: Large-sized test

8 Conclusions

In this chapter the most relevant contributions are presented, together with their relation to the objectives of the thesis. Then, the research lines that arise from this work are envisaged, and finally, the publications that support most of the contributions of this thesis are listed.

8.1 Thesis contributions

The main objective of this PhD thesis has been to investigate and propose optimized techniques for the deployment and scheduling of time-partitioned distributed real-time systems. In order to achieve this objective, several secondary objectives have been proposed, which are fully aligned with the following individual contributions:

- State-of-the-art analysis:

  The deployment and scheduling of distributed real-time systems has been widely addressed in the literature, implementing a wide and diverse range of algorithmic approaches in order to provide schedulable solutions to this NP-hard problem. A considerable number of relevant works have been compiled in several tables in Chapter 2, one for each algorithmic approach, where their most important features have been highlighted. Scheduling and allocation algorithms for partition-based distributed real-time systems are scarce and fairly recent, which is coherent with the current trends in the design of safety critical systems.

- System model formalization:

  A system model that captures the time-related features of a safety critical industrial application has been described and formalized in Chapter 3. It is aligned with OMG's MARTE profile, and it includes all the necessary elements to carry out the response time analysis and scheduling optimization of multipath e2e flows in hierarchically-scheduled systems, as well as in general distributed real-time systems based on fixed priorities.

- Development of a schedulability analysis technique:

  Regarding the schedulability analysis of multipath e2e flows, in Chapter 4 a new method to compute their worst-case response times has been developed. This new technique can be applied not only to time-partitioned systems, but also to general systems scheduled by fixed priorities. It has been shown that the holistic approach, which was the only analysis technique available for multipath e2e flows so far, is a pessimistic approach in comparison to the offset-based approach proposed in this thesis. This technique has been used throughout the thesis to conduct the schedulability analysis in all the proposed optimization algorithms.

- Priority assignment algorithms:

  In Chapter 5 a collection of non-iterative algorithms for priority assignment to multipath e2e flows in hierarchically scheduled and time-partitioned distributed real-time systems has been presented. The method used to adapt state-of-the-art algorithms to multipath e2e flows hosted in hierarchically scheduled architectures is detailed, and they are all applied to the industrial use-case and also to general fixed priority distributed systems. Results show that there is not an algorithm that stands out from the rest, so it makes sense to apply all of them and evaluate which one suits best to the target application, as their execution times are fairly low.

- Partition window assignment algorithm:

  In Chapter 6 the optimization of partition windows is addressed, first by studying the influence of their number and size on the schedulability of the system, and then by proposing an algorithm that leverages this knowledge. The proposed algorithm is based on an algorithm that assigns a number of windows for a fixed utilization in each partition in order to meet the applications' deadlines, and on top of it, another algorithm is in charge of optimizing the utilization of each partition. Results show that this algorithm is capable of finding schedulable solutions for partitioned systems, such as the industrial use-case that motivates this thesis.

- Step-to-processor allocation algorithm:

  Finally, a step-to-processor allocation algorithm is presented in Chapter 7. Although it is not specifically related to time-partitioned systems, which are the main objective of this work, the opportunity of proposing an algorithm that targets heterogeneous multi-processor systems arose during the research stay performed during this thesis. This allocation algorithm is based on the slack, which is a parameter that provides better solutions than the sate-of-the-art allocation algorithms that are based on the processors' utilization.

These contributions constitute a full methodology that comprises modeling, analysis and scheduling optimization phases for developing time-partitioned and distributed real-time systems. Therefore, it can be stated that all the objectives The method used to adapt for this thesis have been successfully fulfilled.

8.2 Future Work

This thesis could form the starting point of many research works. In this section, some of the possible research lines that may be addressed in the near future are listed:

- The proposed response time analysis technique can be optimized by taking the precedence relationships among steps into consideration to reduce the pessimism in the worst-case response time calculations, as it has been done in other offset-based analyses for linear flows.

- As it was stated in Chapter 5, solving the ties produced by some virtual deadline assignment algorithms can be an optimization problem in itself, which should be addressed in order to further tighten applications' worst-case response times.

- Regarding the partition window assignment algorithm proposed in Chapter 6, there are open aspects that it would be appropriate to refine. The first one is related to the assumption of partition window distribution being uniform throughout the MAF, which is a restriction that may be removed in the future in order to optimize the response time of specific partitions. A search and optimization algorithm, such as a genetic algorithm or a simulated annealing algorithm, in combination with the heuristic algorithm developed in Chapter 6, could obtain even more schedulable results.

- Only a small subset of step orderings has been studied in Chapter 7 in order to propose a step-to-processor allocation algorithm. An optimized way of sorting steps in order to minimize the worst-case response times of the target applications may be developed.

- As stated in Chapter 7, slack calculations are independent and therefore they can be parallelized. The proposed allocation algorithm's performance can be greatly improved by making use of a supercomputer that performs such independent calculations in parallel.
- Finally, the methodology developed, which includes modeling, analysis and scheduling optimization phases, can be applied and leveraged in an emergent domain such as the autonomous driving systems based on ROS2, where the real-time performance has been gaining more and more relevance in recent years.

8.3 Publications

Most of the contributions of this thesis have been published in peer-reviewed journals and conferences. Here is a list of these publications:

- **Title:** A review on optimization techniques for the deployment and scheduling of distributed real-time systems
  Authors: A. Amurrio, E. Azketa, J.J. Gutiérrez, M. Aldea, J. Parra
  Journal: Revista Iberoamericana de Automática e Informática Industrial
  Year: 2019
  DOI: 10.4995/riai.2019.10997

- **Title:** Response-Time Analysis of Multipath Flows in Hierarchically-Scheduled Time-Partitioned Distributed Real-Time Systems
  Authors: A. Amurrio, E. Azketa, J.J. Gutiérrez, M. Aldea, M.G. Harbour
  Journal: IEEE Access
  Year: 2020
  DOI: 10.1109/ACCESS.2020.3033461

- **Title:** Priority assignment in hierarchically scheduled time-partitioned distributed real-time systems with multipath flows
  Authors: A. Amurrio, J.J. Gutiérrez, M. Aldea, E. Azketa
  Conference: International Conference on Embedded Software and Systems (ICESS) 2021

Journal: Journal of Systems Architecture
Year: 2021
DOI: 10.1016/j.sysarc.2021.102339

- **Title:** How windows size and number can influence the schedulability of hierarchically-scheduled time-partitioned distributed real-time systems
  Authors: A. Amurrio, J.J. Gutiérrez, M. Aldea, E. Azketa
  Conference: Work-in-progress session at 25th Ada-Europe International Conference on Reliable Software Technologies (AEiC) 2021
  Journal: Ada User Journal
  Year: 2021
  DOI: N/A

The contributions corresponding to Chapters 6 and 7 are also in the ongoing publishing process.

Bibliography

[Aer09] Aeronautical Radio INC. “ARINC Specification 664P7: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network”. In: *AERONAUTICAL RADIO, INC* 2551 (2009), pp. 21401–7435 (cit. on pp. 1, 37).

[ABH10] Ahmad Al Sheikh, Olivier Brun, and Pierre-Emmanuel Hladik. “Partition scheduling on an IMA platform with strict periodicity and communication delays”. In: *18th international conference on real-time and network systems*. 2010, pp. 179–188 (cit. on pp. 23, 25, 29).

[Al +11] Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik, and Balakrishna J Prabhu. “A best-response algorithm for multiprocessor periodic scheduling”. In: *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*. IEEE. 2011, pp. 228–237 (cit. on pp. 29, 32).

[Ali+02] Shoukat Ali, Jong-Kook Kim, Howard Jay Siegel, et al. “Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems.” In: *PDPTA*. 2002, pp. 519–530 (cit. on pp. 27, 32).

[AP04] Luis Almeida and Paulo Pedreiras. “Scheduling within temporal partitions: response-time analysis and server design”. In: *Proceedings of the 4th ACM international conference on Embedded software*. ACM. 2004, pp. 95–103 (cit. on p. 5).

[Alt+12] Ernst Althaus, Sebastian Hoffmann, Joschka Kupilas, and Eike Thaden. “A column generation approach to scheduling of real-time networks”. In: *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1. 2012 (cit. on p. 24).

[Alt+14] Ernst Althaus, Sebastian Hoffmann, Joschka Kupilas, and Eike Thaden. “Scheduling of real-time networks with a column generation approach”. In: *IAENG Transactions on Engineering Technologies*. Springer, 2014, pp. 397–412 (cit. on pp. 24, 25).

[AD94] Rajeev Alur and David L Dill. “A theory of timed automata”. In: *Theoretical computer science* 126.2 (1994), pp. 183–235 (cit. on p. 6).

[Ans+13] Saoussen Anssi, Stefan Kuntz, Sébastien Gérard, and François Terrier. “On the gap between schedulability tests and an automotive task model”. In: *Journal of Systems Architecture* 59.6 (2013), pp. 341–350 (cit. on p. 55).

[Anw+19] Muhammad Waseem Anwar, Muhammad Rashid, Farooque Azam, Muhammad Kashif, and Wasi Haider Butt. “A model-driven framework for design and verification of embedded systems through SystemVerilog”. In: *Design Automation for Embedded Systems* 23.3-4 (2019), pp. 179–223 (cit. on p. 6).

[Anw+20] Muhammad Waseem Anwar, Muhammad Rashid, Farooque Azam, et al. “A Unified Model-Based Framework for the Simplified Execution of Static and Dynamic Assertion-Based Verification”. In: *IEEE Access* 8 (2020), pp. 104407–104431 (cit. on p. 6).

[Aud91] Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report.* Ed. by University of York. 1991 (cit. on p. 30).

[AUT03] AUTOSAR. *AUTomotive Open System ARchitecture*. 2003 (cit. on p. 3).

[Aya+16a] Rabeh Ayari, Imane Hafnaoui, Alexandra Aguiar, et al. “Multi-objective mapping of full-mission simulators on heterogeneous distributed multi-processor systems”. In: *The Journal of Defense Modeling and Simulation* (2016) (cit. on pp. 17, 18).

[Aya+18] Rabeh Ayari, Imane Hafnaoui, Giovanni Beltrame, and Gabriela Nicolescu. “ImGA: an improved genetic algorithm for partitioned scheduling on heterogeneous multi-core systems”. In: *Design Automation for Embedded Systems* (2018), pp. 1–15 (cit. on p. 17).

[Aya+16b] Rabeh Ayari, Imane Hafnaoui, Giovanni Beltrame, and Gabriela Nicolescu. “Schedulability-guided exploration of multi-core systems”. In: *Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. ACM. 2016, pp. 121–127 (cit. on p. 17).

[Azk+11a] Ekain Azketa, J Uribe, Marga Marcos, Luıs Almeida, and J Javier Gutiérrez. “Permutational genetic algorithm for fixed priority scheduling of distributed real-time systems aided by network segmentation”. In: *Proceedings of the 1st Workshop on Synthesis and Optimization Methods for Real-time Embedded Systems*. 2011 (cit. on pp. 16, 18, 32).

[Azk+12] Ekain Azketa, Juan P Uribe, J Javier Gutiérrez, Marga Marcos, and Luıs Almeida. “Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems”. In: *XV Jornadas de Tiempo Real*. 2012 (cit. on p. 16).

[Azk+11b] Ekain Azketa, Juan P Uribe, Marga Marcos, Luis Almeida, and J Javier Gutierrez. “Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems”. In: *Proceedings of the 8th IEEE International Conference on Embedded Software and Systems, pages 958-965*. IEEE. 2011, pp. 958–965 (cit. on p. 16).

[BT18] Clark Barrett and Cesare Tinelli. “Satisfiability modulo theories”. In: *Handbook of Model Checking*. Springer, 2018, pp. 305–343 (cit. on p. 15).

[BBB15] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015 (cit. on pp. 109, 110).

[BLS10] Sanjoy Baruah, Haohan Li, and Leen Stougie. “Towards the design of certifiable mixed-criticality systems”. In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. IEEE. 2010, pp. 13–22 (cit. on p. 3).

[Ber+21] Antoine Bertout, Joël Goossens, Emmanuel Grolleau, Roy Jamil, and Xavier Poczekajlo. “Workload assignment for global real-time scheduling on unrelated clustered platforms”. In: *Real-Time Systems* (2021), pp. 1–32 (cit. on p. 111).

[Ber+20] Antoine Bertout, Joël Goossens, Emmanuel Grolleau, and Xavier Poczekajlo. “Workload assignment for global real-time scheduling on unrelated multicore platforms”. In: *Proceedings of the 28th International Conference on Real-Time Networks and Systems*. 2020, pp. 139–148 (cit. on p. 110).

[BSR17] Anand Bhat, Soheil Samii, and Ragunathan Rajkumar. “Practical task allocation for software fault-tolerance and its implementation in embedded automotive systems”. In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE*. IEEE. 2017, pp. 87–98 (cit. on pp. 31, 32).

[Bli+18] Mathias Blikstad, Emil Karlsson, Tomas Lööw, and Elina Rönnberg. “An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system”. In: *Optimization and Engineering* (2018), pp. 1–28 (cit. on pp. 24, 25).

[Bos91] Robert Bosch Gmbh. “CAN Specification - Version 2.0”. In: (1991) (cit. on p. 3).

[BO14] Fateh Boutekkouk and Soumia Oubadi. “Periodic/Aperiodic tasks scheduling optimization for real time embedded systems with hard/soft constraints”. In: *IT4OD* (2014), p. 135 (cit. on pp. 17, 18).

[BO16] Fateh Boutekkouk and Soumia Oubadi. “Real Time Tasks Scheduling Optimization Using Quantum Inspired Genetic Algorithms”. In: *Artificial Intelligence Perspectives in Intelligent Systems*. Springer, 2016, pp. 69–80 (cit. on p. 17).

[Boy+07] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. “A tutorial on geometric programming”. In: *Optimization and engineering* 8.1 (2007), p. 67 (cit. on p. 14).

[Bra+01] Tracy D Braun, Howard Jay Siegel, Noah Beck, et al. “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems”. In: *Journal of Parallel and Distributed computing* 61.6 (2001), pp. 810–837 (cit. on pp. 26, 32).

[Bra+08] Tracy D Braun, Howard Jay Siegel, Anthony A Maciejewski, and Ye Hong. “Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions”. In: *Journal of Parallel and Distributed Computing* 68.11 (2008), pp. 1504–1516 (cit. on pp. 27, 32).

[BD17] Alan Burns and Robert I Davis. “A survey of research into mixed criticality systems”. In: *ACM Computing Surveys (CSUR)* 50.6 (2017), p. 82 (cit. on pp. 3, 13).

[Bur+93] Alan Burns, Mark Nicholson, K Tindell, and N Zhang. “Allocating and scheduling hard real-time tasks on a point-to-point distributed system”. In: *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*. Citeseer. 1993, pp. 11–20 (cit. on pp. 19, 21).

[Cap+20] Nicola Capodieci, Paolo Burgio, Roberto Cavicchioli, et al. “Real-Time Requirements for ADAS Platforms Featuring Shared Memory Hierarchies”. In: *IEEE Design Test* (2020), pp. 1–1 (cit. on p. 109).

[Cas+18] Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. “Partitioned fixed-priority scheduling of parallel tasks without preemptions”. In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE. 2018, pp. 421–433 (cit. on p. 110).

[Cas+19] Daniel Casini, Paolo Pazzaglia, Alessandro Biondi, Giorgio Buttazzo, and Marco Di Natale. “Addressing analysis and partitioning issues for the waters 2019 challenge”. In: *10th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2019)*. 2019 (cit. on p. 111).

[CDH16] Jinchao Chen, Chenglie Du, and Pengcheng Han. “Scheduling independent partitions in integrated modular avionics systems”. In: *PloS one* 11.12 (2016) (cit. on pp. 30, 32).

[CL00] Wun-Hwa Chen and Chin-Shien Lin. “A hybrid heuristic to solve a task allocation problem”. In: *Computers & Operations Research* 27.3 (2000), pp. 287–303 (cit. on pp. 18, 19).

[CP95] Moreno Coli and Paolo Palazzari. “A new method for optimization of allocation and scheduling in real time applications”. In: *Real-Time Systems, 1995. Proceedings., Seventh Euromicro Workshop on*. IEEE. 1995, pp. 262–269 (cit. on pp. 20, 21).

[CO14] Silviu S Craciunas and Ramon Serna Oliver. “SMT-based task-and network-level static schedule generation for time-triggered networked systems”. In: *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*. ACM. 2014, p. 45 (cit. on pp. 23, 25).

[Cra+16] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelık, and Wilfried Steiner. “Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 183–192 (cit. on p. 23).

[COE14] Silviu S Craciunas, Ramon Serna Oliver, and Valentin Ecker. “Optimal static scheduling of real-time tasks on distributed time-triggered networked systems”. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. IEEE. 2014, pp. 1–8 (cit. on p. 23).

[Cre+14] Alfons Crespo, Alejandro Alonso, Marga Marcos, A Juan, and Patricia Balbastre. “Mixed criticality in control systems”. In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 12261–12271 (cit. on p. 3).

[Dav+07] Abhijit Davare, Qi Zhu, Marco Di Natale, et al. “Period optimization for hard real-time distributed automotive systems”. In: *Proceedings of the 44th annual Design Automation Conference*. ACM. 2007, pp. 278–283 (cit. on pp. 22, 25).

[DB11] Robert I Davis and Alan Burns. “A survey of hard real-time scheduling for multiprocessor systems”. In: *ACM computing surveys (CSUR)* 43.4 (2011), pp. 1–44 (cit. on p. 110).

[DSF17] Emilie Deroche, Jean-Luc Scharbarg, and Christian Fraboul. “A greedy heuristic for distributing hard real-time applications on an IMA architecture”. In: *Industrial Embedded Systems (SIES), 2017 12th IEEE International Symposium on*. IEEE. 2017, pp. 1–8 (cit. on p. 31).

[DSF16] Emilie Deroche, Jean-Luc Scharbarg, and Christian Fraboul. “Mapping real-time communicating tasks on a distributed ima architecture”. In: *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. IEEE. 2016, pp. 1–8 (cit. on pp. 31, 32).

[DS95] Marco Di Natale and John A Stankovic. “Applicability of simulated annealing methods to real-time scheduling and jitter control”. In: *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*. IEEE. 1995, pp. 190–199 (cit. on pp. 20, 21, 32).

[DJ98] Robert P Dick and Niraj K Jha. “MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems”. In: *IEEE transactions on computer-aided design of integrated circuits and systems* 17.10 (1998), pp. 920–935 (cit. on pp. 16, 18).

[DRW98] Robert P Dick, David L Rhodes, and Wayne Wolf. “TGFF: task graphs for free”. In: *Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE’98)*. IEEE. 1998, pp. 97–101 (cit. on pp. 51, 52, 73).

[Eis+10] Friedrich Eisenbrand, Karthikeyan Kesavan, Raju S Mattikalli, et al. “Solving an avionics real-time scheduling problem by advanced IP-methods”. In: *European Symposium on Algorithms*. Springer. 2010, pp. 11–22 (cit. on pp. 29, 32).

[EJ01] Cecilia Ekelin and Jan Jonsson. “Evaluation of search heuristics for embedded system scheduling problems”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2001, pp. 640–654 (cit. on pp. 22, 25).

[Ele+00] Petru Eles, Alex Doboli, Paul Pop, and Zebo Peng. “Scheduling with bus access optimization for distributed embedded systems”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8.5 (2000), pp. 472–491 (cit. on pp. 27, 32).

[EB10] Paul Emberson and Iain Bate. “Stressing search with scenarios for flexible solutions to real-time task allocation problems”. In: *IEEE Transactions on Software Engineering* 36.5 (2010), pp. 704–718 (cit. on p. 20).

[ERT06] ERTMS/ETCS. “European Rail Traffic Management System/European Train Control System release notes to system requirements specification”. In: *Subset 026 version 2.3.0* (2006) (cit. on p. 7).

[FDB00] Sebastien Faucou, A-M Deplanche, and J-P Beauvais. “Heuristic techniques for allocating and scheduling communicating periodic tasks in distributed real-time systems”. In: *Factory Communication Systems, 2000. Proceedings. 2000 IEEE International Workshop on*. IEEE. 2000, pp. 257–265 (cit. on pp. 16, 18).

[FGB10] Nathan Fisher, Joël Goossens, and Sanjoy Baruah. “Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible”. In: *Real-Time Systems* 45.1 (2010), pp. 26–71 (cit. on p. 109).

[FF98] Carlos M Fonseca and Peter J Fleming. “Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation”. In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 28.1 (1998), pp. 26–37 (cit. on p. 16).

[Fon+16] José Fonseca, Geoffrey Nelissen, Vincent Nelis, and Luıs Miguel Pinho. “Response time analysis of sporadic dag tasks under partitioned scheduling”. In: *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*. IEEE. 2016, pp. 1–10 (cit. on p. 5).

[Gar+14] Ricardo Garibay Martínez, Geoffrey Nelissen, Luis Lino Ferreira, and Luis Miguel Pinho. “On the scheduling of fork-join parallel/distributed real-time tasks”. In: *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*. IEEE. 2014, pp. 31–40 (cit. on p. 30).

[Gar+15] Ricardo Garibay-Martınez, Geoffrey Nelissen, Luis Lino Ferreira, and Luis Miguel Pinho. “Task partitioning and priority assignment for distributed hard real-time systems”. In: *Journal of Computer and System Sciences* 81.8 (2015), pp. 1542–1555 (cit. on pp. 30, 32, 110).

[Gre] Green Hills Software. *Integrity RTOS* (cit. on p. 3).

[Glo86] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5 (1986), pp. 533–549 (cit. on p. 14).

[GH88] David E Goldberg and John H Holland. “Genetic algorithms and machine learning”. In: *Machine learning* 3.2 (1988), pp. 95–99 (cit. on p. 16).

[Gon+01] Michael González Harbour, J Javier Gutiérrez, J Carlos Palencia, and J Maria Drake. “Mast: Modeling and analysis suite for real time applications”. In: *in Proceedings of the 13th Euromicro Conference on Real-Time Systems*. IEEE. 2001, pp. 125–134 (cit. on pp. 4, 35, 46).

[Goo+13] Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, et al. “Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow”. In: *ACM SIGBED Review* 10.3 (2013), pp. 23–34 (cit. on p. 3).

[Gua+20] Ana Guasque, Hossein Tohidi, Patricia Balbastre, et al. “Integer Programming Techniques for Static Scheduling of Hard Real-Time Systems”. In: *IEEE Access* 8 (2020), pp. 170389–170403 (cit. on pp. 24, 25).

[GG95] J Javier Gutiérrez and M González Harbour. “Optimized priority assignment for tasks and messages in distributed hard real-time systems”. In: *Proceedings of Third Workshop on Parallel and Distributed Real-Time Systems*. IEEE. 1995, pp. 124–132 (cit. on pp. 16, 26, 32).

[GPH14] J Javier Gutiérrez, J Carlos Palencia, and Michael González Harbour. “Holistic schedulability analysis for multipacket messages in AFDX networks”. In: *Real-Time Systems* 50.2 (2014), pp. 230–269 (cit. on p. 37).

[GPH00] J. Javier Gutiérrez, J. Carlos Palencia, and Michael González Harbour. “Schedulability analysis of distributed hard real-time systems with multiple-event synchronization”. In: *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*. IEEE. 2000, pp. 15–24 (cit. on pp. 5, 49, 51).

[Ham+06] Arne Hamann, Marek Jersak, Kai Richter, and Rolf Ernst. “A framework for modular analysis and exploration of heterogeneous embedded systems”. In: *Real-Time Systems* 33.1-3 (2006), pp. 101–137 (cit. on pp. 16, 18, 29).

[Ham+] Etienne Hamelin, Moha Ait Hmid, Amine Naji, and Yves Mouafo-Tchinda. “Selection and evaluation of an embedded hypervisor: application to an automotive platform”. In: (). Proceedings of the 10th Embedded Real-Time Systems International Congress (ERTS 2020) (cit. on p. 83).

[HZZ20] Pujie Han, Zhengjun Zhai, and Lei Zhang. “A Model-Based Approach to Optimizing Partition Scheduling of Integrated Modular Avionics Systems”. In: *Electronics* 9.8 (2020), p. 1281 (cit. on p. 6).

[Har+13] Michael González Harbour, J Javier Gutiérrez, José M Drake, Patricia López, and J Carlos Palencia. “Modeling distributed real-time systems with MAST 2”. In: *Journal of Systems Architecture* 59.6 (2013), pp. 331–340 (cit. on pp. 4, 35).

[Har82] Juris Hartmanis. “Computers and intractability: a guide to the theory of NP-completeness (michael r. garey and david s. johnson)”. In: *Siam Review* 24.1 (1982), p. 90 (cit. on p. 109).

[HGZ10] Xiuqiang He, Zonghua Gu, and Yongxin Zhu. “Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming”. In: *The Computer Journal* 53.7 (2010), pp. 1071–1091 (cit. on pp. 20, 21, 32).

[HE05] Rafik Henia and Rolf Ernst. “Context-aware scheduling analysis of distributed systems with tree-shaped task-dependencies”. In: *Design, Automation and Test in Europe*. IEEE. 2005, pp. 480–485 (cit. on p. 5).

[Hen+15] Rafik Henia, Laurent Rioux, Nicolas Sordon, Gérald-Emmanuel Garcia, and Marco Panunzio. "Integrating Formal Timing Analysis in the Real-Time Software Development Process". In: *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*. 2015, pp. 35–40 (cit. on p. 4).

[Hes+08] Anders Hessel, Kim G Larsen, Marius Mikucionis, et al. "Testing real-time systems using UPPAAL". In: *Formal methods and testing*. Springer, 2008, pp. 77–117 (cit. on p. 6).

[Hla+08] Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. "Solving a real-time allocation problem with constraint programming". In: *Journal of Systems and Software* 81.1 (2008), pp. 132–149 (cit. on pp. 22, 25).

[Hol75] John Holland. "Adaptation in artificial and natural systems". In: *Ann Arbor: The University of Michigan Press* (1975) (cit. on p. 14).

[Höt+19] Robert Höttger, Junhyung Ki, Burkhard Igel, Olaf Spinczyk, et al. "Cpu-gpu response time and mapping analysis for highperformance automotive systems". In: *10th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2019)*. 2019 (cit. on p. 111).

[HS97] Chao-Ju Hou and Kang G. Shin. "Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems". In: *IEEE transactions on computers* 46.12 (1997), pp. 1338–1356 (cit. on pp. 25, 26).

[HAR94] Edwin SH Hou, Nirwan Ansari, and Hong Ren. "A genetic algorithm for multiprocessor scheduling". In: *IEEE Transactions on Parallel and Distributed systems* 5.2 (1994), pp. 113–120 (cit. on pp. 15, 18).

[Hu+15] Menglan Hu, Jun Luo, Yang Wang, and Bharadwaj Veeravalli. "Scheduling periodic task graphs for safety-critical time-triggered avionic systems." In: *IEEE Trans. Aerospace and Electronic Systems* 51.3 (2015), pp. 2294–2304 (cit. on pp. 30, 32).

[IEC10] IEC. "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems Part 1: General requirements". In: (2010) (cit. on p. 7).

[IEE03] Pasc IEEE Portable Application Standards Committee. *Standard for Information Technology-Portable Operating System Interface (POSIX) Realtime and Embedded Application Support. Std. 1003.13.* 2003 (cit. on p. 3).

[ISO12] ISO/IEC. *Ada 2012 Reference Manual. Language and Standard Libraries - Interna-tional Standard ISO/IEC 8652:2012(E)*. 2012 (cit. on p. 3).

[JPJ17] Wei Jiang, Paul Pop, and Ke Jiang. "Design optimization for security-and safety-critical distributed real-time applications". In: *Microprocessors and Microsystems* 52 (2017), pp. 401–415 (cit. on p. 19).

[Joh99] Rushby John. “Partitioning in avionics architectures: requirements, mechanisms, and assurance”. In: (1999) (cit. on p. 23).

[KG93a] H Kao and Hector Garcia-Molina. “Deadline assignment in a distributed soft real-time system”. In: *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*. IEEE. 1993, pp. 428–437 (cit. on p. 59).

[Kir84] Scott Kirkpatrick. “Optimization by simulated annealing: Quantitative studies”. In: *Journal of statistical physics* 34.5-6 (1984), pp. 975–986 (cit. on p. 14).

[Klo+13] Kay Klobedanz, Jan Jatzkowski, Achim Rettberg, and Wolfgang Mueller. “Fault-tolerant deployment of real-time software in AUTOSAR ECU networks”. In: *International Embedded Systems Symposium*. Springer. 2013, pp. 238–249 (cit. on pp. 29, 32).

[Kop11] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011 (cit. on p. 2).

[KG93b] Hermann Kopetz and Günter Grunsteidl. “TTP-A time-triggered protocol for fault-tolerant real-time systems”. In: *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*. IEEE. 1993, pp. 524–533 (cit. on p. 20).

[KHB16] Philip S Kurtin, Joost PHM Hausmans, and Marco JG Bekooij. “Combining offsets with precedence constraints to improve temporal analysis of cyclic real-time streaming applications”. In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2016, pp. 1–12 (cit. on p. 6).

[LD60] Ailsa H Land and Alison G Doig. “An automatic method of solving discrete programming problems”. In: *Econometrica: Journal of the Econometric Society* (1960), pp. 497–520 (cit. on p. 15).

[LSD89] John Lehoczky, Lui Sha, and Yuqin Ding. “The rate monotonic scheduling algorithm: Exact characterization and average case behavior”. In: *RTSS*. Vol. 89. 1989, pp. 166–171 (cit. on p. 76).

[LKY00] Man Lin, Lars Karlsson, and Laurence Tianruo Yang. “Heuristic techniques: Scheduling partially ordered tasks in a multi-processor environment with tabu search and genetic algorithms”. In: *Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000*. IEEE. 2000, pp. 515–523 (cit. on pp. 18, 19).

[LA11] Cong Liu and James H Anderson. “Supporting graph-based real-time applications in distributed systems”. In: *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*. Vol. 1. IEEE. 2011, pp. 143–152 (cit. on p. 5).

[LA10] Cong Liu and James H Anderson. “Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss”. In: *2010 31st IEEE Real-Time Systems Symposium*. IEEE. 2010, pp. 3–13 (cit. on pp. 5, 114).

[Liu00] JWS Liu. “Real-Time Systems”. In: *Prentice Hall* 48 (2000), p. 42 (cit. on pp. 3, 58, 59, 62).

[MN08] Jukka Mäki-Turja and Mikael Nolin. “Efficient implementation of tight response-times for tasks with offsets”. In: *Real-Time Systems* 40.1 (2008), pp. 77–116 (cit. on pp. 5, 48, 49, 51).

[Mar+12] Sorin Ovidiu Marinescu, Domiţian Tămaş-Selicean, Vlad Acretoaie, and Paul Pop. “Timing analysis of mixed-criticality hard real-time applications implemented on distributed partitioned architectures”. In: *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. IEEE. 2012, pp. 1–4 (cit. on p. 6).

[Mar03] Rafael Martı. “Multi-start methods”. In: *Handbook of metaheuristics*. Springer, 2003, pp. 355–368 (cit. on p. 20).

[Mas+] Miguel Masmano, Ismael Ripoll, Alfons Crespo, and J Metge. “Xtratum: a hypervisor for safety critical embedded systems”. In: Proceedings of the 11th Real-Time Linux Workshop 2009, pages 263-272 (cit. on p. 83).

[McL+20] Shane D McLean, Silviu S Craciunas, Emil Alexander Juul Hansen, and Paul Pop. “Mapping and scheduling automotive applications on ADAS platforms using metaheuristics”. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE. 2020, pp. 329–336 (cit. on p. 21).

[Meh+13] Asma Mehiaoui, Ernest Wozniak, Sara Tucci-Piergiovanni, et al. “A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems”. In: *ACM SIGPLAN Notices* 48.5 (2013), pp. 121–132 (cit. on pp. 29, 32).

[Mel+15] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C Buttazzo. “Response-time analysis of conditional dag tasks in multiprocessor systems”. In: *2015 27th Euromicro Conference on Real-Time Systems*. IEEE. 2015, pp. 211–221 (cit. on p. 114).

[MH06] Alexander Metzner and Christian Herde. “Rtsat–an optimal and efficient approach to the task allocation problem in distributed architectures”. In: *Real-Time Systems Symposium, 2006. RTSS’06. 27th IEEE International*. IEEE. 2006, pp. 147–158 (cit. on pp. 22, 25).

[Min+18] Anna Minaeva, Benny Akesson, Zdeněk Hanzálek, and Dakshina Dasari. “Time-triggered co-scheduling of computation and communication with jitter requirements”. In: *IEEE Transactions on Computers* 67.1 (2018), pp. 115–129 (cit. on pp. 24, 25).

[Min86] Michel Minoux. *Mathematical programming: theory and algorithms*. John Wiley & Sons, 1986 (cit. on p. 14).

[MR93] Hirak Mitra and Parameswaran Ramanathan. “A genetic approach for scheduling non-preemptive tasks with precedence and deadline constraints”. In: *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*. Vol. 2. IEEE. 1993, pp. 556–564 (cit. on pp. 15, 18).

[MBD98] Yannick Monnier, J-P Beauvais, and A-M Deplanche. “A genetic algorithm for scheduling tasks in a real-time distributed system”. In: *Euromicro Conference, 1998. Proceedings. 24th*. Vol. 2. IEEE. 1998, pp. 708–714 (cit. on pp. 15, 18).

[NSE11] Moritz Neukirchner, Steffen Stein, and Rolf Ernst. “A lazy algorithm for distributed priority assignment in real-time systems”. In: *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*. IEEE. 2011, pp. 126–132 (cit. on pp. 29, 32).

[Obj11] Object Management Group. “UML profile for MARTE: Modeling and Analysis of Real Time Embedded Systems, version 1.1.” In: *OMG Document Formal* (2011) (cit. on pp. 4, 35).

[OW04] Jaewon Oh and Chisu Wu. “Genetic-algorithm-based real-time task scheduling with multiple goals”. In: *Journal of systems and software* 71.3 (2004), pp. 245–258 (cit. on pp. 16, 18).

[OCC18] Ignacio Sañudo Olmedo, Nicola Capodieci, and Roberto Cavicchioli. “A Perspective on Safety and Real-Time Issues for GPU Accelerated ADAS”. In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. 2018, pp. 4071–4077 (cit. on p. 110).

[Pal+16] J Carlos Palencia, Michael González Harbour, J Javier Gutiérrez, and Juan M Rivas. “Response-time analysis in hierarchically-scheduled time-partitioned distributed systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.7 (2016), pp. 2017–2030 (cit. on pp. 5, 45, 48).

[PG98] Jose Carlos Palencia and Michael González Harbour. “Schedulability analysis for tasks with static and dynamic offsets”. In: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE. 1998, pp. 26–37 (cit. on pp. 5, 49, 53).

[PG99] José Carlos Palencia and Michael González Harbour. “Exploiting precedence relations in the schedulability analysis of distributed real-time systems”. In: *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054)*. IEEE. 1999, pp. 328–339 (cit. on p. 5).

[Pea84] Judea Pearl. “Heuristics: intelligent search strategies for computer problem solving”. In: *Addison-Wesley Pub. Co., Inc., Reading, MA* (1984) (cit. on p. 15).

[PA04] Paulo Pedreiras and Luis Almeida. “Message routing in multi-segment FTT networks: The Isochronous Approach”. In: *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE. 2004, p. 122 (cit. on pp. 28, 32).

[PSA97] Dar-Tzen Peng, Kang G. Shin, and Tarek F. Abdelzaher. “Assignment and scheduling communicating periodic tasks in distributed real-time systems”. In: *IEEE Transactions on Software Engineering* 23.12 (1997), pp. 745–758 (cit. on pp. 25, 26).

[Per+09] Simon Perathoner, Ernesto Wandeler, Lothar Thiele, et al. "Influence of different abstractions on the performance analysis of distributed hard real-time systems". In: *Design Automation for Embedded Systems* 13.1-2 (2009), pp. 27–49 (cit. on p. 6).

[PA15] Mohammad Amin Pishdar and Abbas Akkasi. "Task Scheduling and Idle-Time Balancing in Homogeneous Multi Processors: A Comparison between GA and SA". In: *International Journal of Computer Applications* 123.13 (2015) (cit. on p. 21).

[PEP04a] Paul Pop, Petru Eles, and Zebo Peng. *Analysis and synthesis of distributed real-time embedded systems*. Springer Science & Business Media, 2004 (cit. on p. 3).

[PEP00] Paul Pop, Petru Eles, and Zebo Peng. "Bus access optimization for distributed embedded systems based on schedulability analysis". In: *Proceedings of the conference on Design, automation and test in Europe*. ACM. 2000, pp. 567–575 (cit. on pp. 27, 32).

[PEP04b] Paul Pop, Petru Eles, and Zebo Peng. "Schedulability-driven communication synthesis for time triggered embedded systems". In: *Real-Time Systems* 26.3 (2004), pp. 297–325 (cit. on p. 27).

[Pop+04] Paul Pop, Petru Eles, Zebo Peng, and Viacheslav Izosimov. "Schedulability-driven partitioning and mapping for multi-cluster real-time systems". In: *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. IEEE. 2004, pp. 91–100 (cit. on pp. 27, 32).

[Pop+01a] Paul Pop, Petru Eles, Traian Pop, and Zebo Peng. "An approach to incremental design of distributed embedded systems". In: *Proceedings of the 38th annual Design Automation Conference*. ACM. 2001, pp. 450–455 (cit. on pp. 27, 32).

[Pop+01b] Paul Pop, Petru Eles, Traian Pop, and Zebo Peng. "Minimizing system modification in an incremental design approach". In: *Proceedings of the ninth international symposium on Hardware/software codesign*. ACM. 2001, pp. 183–188 (cit. on p. 27).

[Pop07] Traian Pop. "Analysis and optimisation of distributed embedded systems with heterogeneous scheduling policies". PhD thesis. Institutionen för datavetenskap, 2007 (cit. on pp. 28, 32).

[PEP03a] Traian Pop, Petru Eles, and Zebo Peng. "Design optimization of mixed time/event-triggered distributed embedded systems". In: *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM. 2003, pp. 83–89 (cit. on p. 28).

[PEP03b] Traian Pop, Petru Eles, and Zebo Peng. "Schedulability analysis for distributed heterogeneous time/event triggered real-time systems". In: *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*. IEEE. 2003, pp. 257–266 (cit. on p. 27).

[Pop+05] Traian Pop, Paul Pop, Petru Eles, and Zebo Peng. "Optimization of hierarchically scheduled heterogeneous embedded systems". In: *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*. IEEE. 2005, pp. 67–71 (cit. on p. 28).

[PKR00] Stella CS Porto, João Paulo FW Kitajima, and Celso C Ribeiro. "Performance evaluation of a parallel tabu search task scheduling algorithm". In: *Parallel Computing* 26.1 (2000), pp. 73–90 (cit. on pp. 18, 19).

[QJ06] Xiao Qin and Hong Jiang. "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems". In: *Parallel Computing* 32.5-6 (2006), pp. 331–356 (cit. on pp. 28, 32).

[Ram95] Krithi Ramamritham. "Allocation and scheduling of precedence-related periodic tasks". In: *IEEE Transactions on Parallel and Distributed Systems* 6.4 (1995), pp. 412–420 (cit. on pp. 26, 32).

[Reh+21] Falk Rehm, Dakshina Dasari, Arne Hamann, et al. "Performance modeling of heterogeneous HW platforms". In: *Microprocessors and Microsystems* 87 (2021), p. 104336 (cit. on p. 111).

[RRC03] Michael Richard, Pascal Richard, and Francis Cottet. "Allocating and scheduling tasks in multiple fieldbus real-time systems". In: *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*. Vol. 1. IEEE. 2003, pp. 137–144 (cit. on pp. 25, 26).

[Riv+14] Juan M Rivas, J Javier Gutierrez, J Carlos Palencia, and Michael Gonzalez Harbour. "Deadline assignment in EDF schedulers for real-time distributed systems". In: *IEEE Transactions on Parallel and Distributed Systems* 26.10 (2014), pp. 2671–2684 (cit. on pp. 57–59, 62).

[RGH17] Juan M Rivas, J Javier Gutiérrez, and Michael González Harbour. "A supercomputing framework for the evaluation of real-time analysis and optimization techniques". In: *Journal of Systems and Software* 124 (2017), pp. 120–136 (cit. on p. 114).

[Riv+11] Juan M Rivas, J Javier Gutiérrez, J Carlos Palencia, Michael González Harbour, et al. "Schedulability analysis and optimization of heterogeneous edf and fp distributed real-time systems". In: *2011 23rd Euromicro Conference on Real-Time Systems*. IEEE. 2011, pp. 195–204 (cit. on pp. 5, 37, 45).

[RG+16] Juan Marıa Rivas Concepción, José Javier Gutiérrez Garcıa, et al. "Interpretación de dos algoritmos EDF on-line para la optimización de sistemas distribuidos de tiempo real". In: *(In Spanish)* (2016) (cit. on pp. 57, 59, 60, 65).

[Sam+09] Soheil Samii, Yanfei Yin, Zebo Peng, Petru Eles, and Yuanping Zhang. "Immune genetic algorithms for optimization of task priorities and FlexRay frame identifiers". In: *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09. 15th IEEE International Conference on*. IEEE. 2009, pp. 486–493 (cit. on pp. 16, 18).

[SD07] Alberto Sangiovanni-Vincentelli and Marco Di Natale. “Embedded system design for automotive applications”. In: *Computer* 40.10 (2007), pp. 42–51 (cit. on p. 73).

[Sch06] Douglas C Schmidt. “Model-driven engineering”. In: *Computer-IEEE Computer Society-* 39.2 (2006), p. 25 (cit. on p. 3).

[Sch98] Alexander Schrijver. “Theory of linear and integer programming”. In: *Wiley* (1998) (cit. on pp. 14, 23).

[SLB10] Nicola Serreli, Giuseppe Lipari, and Enrico Bini. “The distributed deadline synchronization protocol for real-time systems scheduled by EDF”. In: *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*. IEEE. 2010, pp. 1–8 (cit. on p. 58).

[SDJ07] Li Shang, Robert P Dick, and Niraj K Jha. “Slopes: hardware–software cosynthesis of low-power real-time distributed embedded systems with dynamically reconfigurable fpgas”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.3 (2007), pp. 508–526 (cit. on pp. 16, 18).

[SS04] David John Smith and Kenneth GL Simpson. *Functional Safety: A straightforward guide to applying IEC 61508 and related standards*. Routledge, 2004 (cit. on p. 2).

[SO12] European Commission. Information Society and Media Directorate-General Unit G3/Computing Systems Research Objective. “Mixed Criticality Systems”. In: *Report from the Workshop on Mixed Criticality Systems* (2012) (cit. on p. 3).

[SK03] Radoslaw Szymanek and Krzysztof Krzysztof. “Partial task assignment of task graphs under heterogeneous resource constraints”. In: *Proceedings of the 40th annual Design Automation Conference*. ACM. 2003, pp. 244–249 (cit. on p. 21).

[SK01] Radoslaw Szymanek and Krzysztof Kuchcinski. “A constructive algorithm for memory-aware task assignment and scheduling”. In: *Proceedings of the ninth international symposium on Hardware/software codesign*. ACM. 2001, pp. 147–152 (cit. on pp. 21, 25).

[TP15] Domiţian Tămaş-Selicean and Paul Pop. “Design optimization of mixed-criticality real-time embedded systems”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 14.3 (2015), p. 50 (cit. on p. 19).

[TP11a] Domiţian Tămaş-Selicean and Paul Pop. “Optimization of time-partitions for mixed-criticality real-time distributed embedded systems”. In: *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*. IEEE. 2011, pp. 1–10 (cit. on pp. 20, 21).

[TP11b] Domiţian Tămaş-Selicean and Paul Pop. “Task mapping and partition allocation for mixed-criticality real-time systems”. In: *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*. IEEE. 2011, pp. 282–283 (cit. on pp. 18, 19).

[Tin94] Ken Tindell. *Adding time-offsets to schedulability analysis*. Technical Report, 1994 (cit. on pp. 5, 49).

[TC94] Ken Tindell and John Clark. "Holistic schedulability analysis for distributed hard real-time systems". In: *Microprocessing and microprogramming* 40.2-3 (1994), pp. 117–134 (cit. on pp. 4, 28).

[TBW92] Ken W Tindell, Alan Burns, and Andy J. Wellings. "Allocating hard real-time tasks: an NP-hard problem made easy". In: *Real-Time Systems* 4.2 (1992), pp. 145–165 (cit. on pp. 2, 13, 19, 21, 57).

[Tru+14] Salvador Trujillo, Alfons Crespo, Alejandro Alonso, and Jon Pérez. "MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems". In: *Microprocessors and Microsystems* 38.8 (2014), pp. 921–932 (cit. on p. 3).

[Tsa14] Edward Tsang. *Foundations of constraint satisfaction: the classic text*. BoD–Books on Demand, 2014 (cit. on p. 14).

[TSN] Time Sensitive Networking Task Group TSN. "IEEE-802.1 Standard". In: () (cit. on p. 1).

[UNI15] UNISIG. " ERTMS/ETCS -SUBSET-041- Performance Requirements for Interoperability". In: (2015) (cit. on p. 49).

[VO05] Lucy Marıa Franco Vargas and Romulo Silva de Oliveira. "Empirical study of tabu search, simulated annealing and multi-start in fieldbus scheduling". In: *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*. Vol. 2. IEEE. 2005, 8–pp (cit. on pp. 20, 21).

[VP08] Ian Verhappen and Augusto Pereira. *Foundation Fieldbus*. ISA, 2008 (cit. on p. 20).

[Ver+20] Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. "Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets". In: *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2020, pp. 226–238 (cit. on p. 114).

[Ves07] Steve Vestal. "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance". In: *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE. 2007, pp. 239–243 (cit. on p. 3).

[Win16] WindRiver. *Wind River VxWorks 653 Platform*. 2016 (cit. on p. 3).

[WME20] Brandon Woolley, Susan Mengel, and Atila Ertas. "An Evolutionary Approach for the Hierarchical Scheduling of Safety-and Security-Critical Multicore Architectures". In: *Computers* 9.3 (2020), p. 71 (cit. on pp. 17, 18).

[Woz+13] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, and Sébastien Gerard. "An optimization approach for the synthesis of AUTOSAR architectures". In: *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE. 2013, pp. 1–10 (cit. on pp. 17, 18, 29).

[Wur+19] Falk Wurst, Dakshina Dasari, Arne Hamann, et al. “System performance modelling of heterogeneous HW platforms: An automated driving case study”. In: *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE. 2019, pp. 365–372 (cit. on p. 111).

[Xie+16] Guoqi Xie, Gang Zeng, Liangjiao Liu, Renfa Li, and Keqin Li. “High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems”. In: *Journal of Systems Architecture* 70 (2016), pp. 3–14 (cit. on pp. 30, 32).

[Yoo09] Myungryun Yoo. “Real-time task scheduling by multiobjective genetic algorithm”. In: *Journal of Systems and Software* 82.4 (2009), pp. 619–628 (cit. on pp. 16, 18).

[YR15] H Yoon and M Ryu. “Guaranteeing end-to-end deadlines for AUTOSAR-based automotive software”. In: *International Journal of Automotive Technology* 16.4 (2015), pp. 635–644 (cit. on pp. 30, 32).

[Zha+14] Licong Zhang, Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. “Task-and network-level schedule co-synthesis of Ethernet-based time-triggered systems”. In: *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE. 2014, pp. 119–124 (cit. on pp. 23, 25).

[ZZ19] Yecheng Zhao and Haibo Zeng. “The concept of Maximal Unschedulable Deadline Assignment for optimization in fixed-priority scheduled real-time systems”. In: *Real-Time Systems* 55.3 (2019), pp. 667–707 (cit. on pp. 31, 32).

[Zhe+07a] Wei Zheng, Marco Di Natale, Claudio Pinello, Paolo Giusto, and Alberto Sangiovanni Vincentelli. “Synthesis of task and message activation models in real-time distributed automotive systems”. In: *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE’07*. IEEE. 2007, pp. 1–6 (cit. on pp. 22, 25).

[Zhe+07b] Wei Zheng, Qi Zhu, Marco Di Natale, and Alberto Sangiovanni Vincentelli. “Definition of task allocation and priority assignment in hard real-time distributed systems”. In: *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE. 2007, pp. 161–170 (cit. on pp. 22, 25).

[Zho+19] Yuanbin Zhou, Soheil Samii, Petru Eles, and Zebo Peng. “Partitioned and overhead-aware scheduling of mixed-criticality real-time systems”. In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 2019, pp. 39–44 (cit. on pp. 31, 32).

[Zhu+09] Qi Zhu, Yang Yang, Eelco Scholte, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. “Optimizing extensibility in hard real-time distributed systems”. In: *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*. IEEE. 2009, pp. 275–284 (cit. on pp. 23, 25).

[Zhu+12] Qi Zhu, Haibo Zeng, Wei Zheng, Marco DI Natale, and Alberto Sangiovanni-Vincentelli. "Optimization of task allocation and priority assignment in hard real-time distributed systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 11.4 (2012), p. 85 (cit. on pp. 23, 25).

List of Figures

1.1 Architecture of EB functionality 8

3.1 Distributed multipath e2e flow 36
3.2 Example of hierarchical scheduler 39
3.3 Partition and effective partition with $CS_y = 1ms$ 39
3.4 Industrial use-case modeling (RBC-CS & PV-DMI not depicted for the sake of clarity) 43

4.1 Simple example 48
4.2 Synthetic application generated with TGFF 52
4.3 Differences between worst-case response times obtained with the holistic and the offset-based analysis 54

5.1 Illustrative example 61
5.2 Synthetic e2e flows 75
5.3 RBU evolution in 1 processor 78
5.4 RBU evolution in 2 processors 78
5.5 RBU evolution in 4 processors 78
5.6 RBU evolution in 8 processors 79
5.7 RBU evolution in 10 processors 79

6.1 Guiding application example 82
6.2 Worst-case response time of $\tau_{1\ 6}$ as a function of AU_{Px} 82
6.3 Evolution of worst-case response time of $\tau_{1\ 6}$ with the number of partition windows for different values of AU_{P_x} (in %) 83
6.4 Worst-case response time as a function of the number of partition windows - $AU_{P_x} = 40\%$ 84
6.5 Worst-case response time as a function of the number of partition windows - $AU_{P_x} = 50\%$ 85
6.6 Worst-case response time as a function of the number of partition windows - $AU_{P_x} = 60\%$ 86
6.7 Baseline synthetic scenario **Scn1**: 2 partitions hosted in each of the 4 processors 93

6.8 Scn2: Γ_1 has been removed from CPU_1 and its steps relocated to partitions in the other processors 93
6.9 Scn3: Γ_2 has also been removed from CPU_1 and its steps relocated to partitions in the other processors 94
6.10 e2e flows' worst-case response times for different numbers of windows (Scn1) 96
6.11 e2e flows' worst-case response times for different numbers of windows (Scn2) 97
6.12 e2e flows' worst-case response times for different numbers of windows (Scn3) 97
6.13 e2e flows' worst-case response times for different numbers of windows (Scn4) 98
6.14 e2e flows' worst-case response times for different numbers of windows (Scn5) 98

7.1 Small-sized test 116
7.2 Medium-sized test 118
7.3 Large-sized test 119

List of Tables

2.1 Reviewed Works - Genetic Algorithm 18

2.2 Reviewed Works - Tabu Search 19

2.3 Reviewed Works - Simulated Annealing 21

2.4 Reviewed Works - Mathematical Programming 25

2.5 Reviewed Works - Branch and Bound 26

2.6 Reviewed Works - Heuristics 32

3.1 Summary of notation 41

3.2 Train signalling application (times in μs) 43

4.1 Analysis results of the simple example (times in ms) 48

4.2 Response-time analysis of a train signalling application (times in μs) . 50

4.3 Synthetic application analysis, times in ms. *(Hol) = Holistic Analysis, (Off) = Offset-based analysis* 53

5.1 Virtual Deadlines, priorities and worst-case response times for each algorithm, applied to the multipath e2e flow depicted in Figure 5.1 . . 69

5.2 Priority assignment for the train signalling application (C_{ij} in μs) . . . 71

5.3 Worst-case response times of the railway signalling application 72

5.4 SSF for each algorithm applied in the industrial use-case 72

5.5 Step-to-processor mapping of the synthetic e2e flows 74

5.6 RBU results (in %) for each algorithm (with partitioning) 76

5.7 BU results (in %) for each algorithm 77

5.8 Execution time of some of the experiments 80

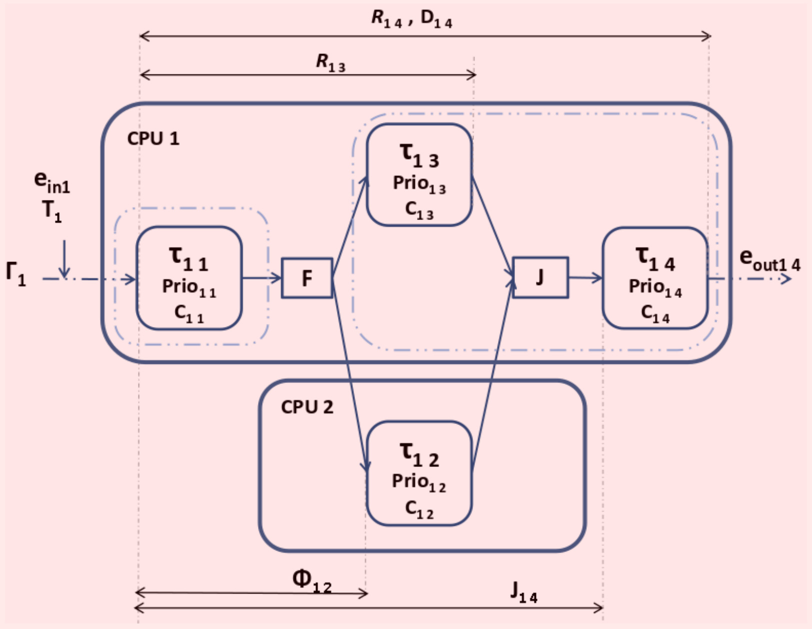
6.1 Worst-case execution times for Scn1 94

6.2 Schedulable solutions found by WinAs: MAF and worst-case response times 99

6.3 Initial available utilization (in %) for each partition in each partition configuration 99

6.4 MAF and AU values obtained by HOPWA, for each partition configuration in Scn1 100

6.5 MAF and AU values obtained by HOPWA, for each partition configuration in Scn2 . . . 100
6.6 MAF and AU values obtained by HOPWA, for each partition configuration in Scn3 . . . 100
6.7 MAF and AU values obtained by HOPWA, for each partition configuration in Scn4 . . . 101
6.8 MAF and AU values obtained by HOPWA, for each partition configuration in Scn5 . . . 101
6.9 Slack factors (system, partitions and e2e flows) for Scn1 . . . 103
6.10 Slack factors (system, partitions and e2e flows) for Scn2 . . . 104
6.11 Slack factors (system, partitions and e2e flows) for Scn3 . . . 104
6.12 Slack factors (system, partitions and e2e flows) for Scn4 . . . 105
6.13 Slack factors (system, partitions and e2e flows) for Scn5 . . . 105
6.14 Initial available utilization (in %) for each partition in each Configuration (Config.) . . . 106
6.15 MAF and AU values obtained by HOPWA for the railway use-case . . . 107
6.16 Worst-case response times (in ms) of the railway application under the different configurations and deadlines tested . . . 107



$R_{1\,4}$, $D_{1\,4}$
$R_{1\,3}$
CPU 1
e_{in1}
T_1
Γ_1
$\tau_{1\,1}$ $Prio_{1\,1}$ $C_{1\,1}$
F
$\tau_{1\,3}$ $Prio_{1\,3}$ $C_{1\,3}$
J
$\tau_{1\,4}$ $Prio_{1\,4}$ $C_{1\,4}$
$e_{out1\,4}$
CPU 2
$\tau_{1\,2}$ $Prio_{1\,2}$ $C_{1\,2}$
$\Phi_{1\,2}$
$J_{1\,4}$

La creciente complejidad de los sistemas de control modernos lleva a muchas empresas a tener que re-dimensionar o re-diseñar sus soluciones para adecuarlas a nuevas funcionalidades y requisitos. Un caso paradigmático de esta situación se ha dado en el sector ferroviario, donde la implementación de las aplicaciones de señalización se ha llevado a cabo empleando técnicas tradicionales que, si bien ahora mismo cumplen con los requisitos básicos, su rendimiento temporal y escalabilidad funcional son sustancialmente mejorables. A partir de las soluciones propuestas en esta tesis, además de contribuir a la validación de sistemas que requieren certificación de seguridad funcional, también se creará la tecnología base de análisis de planificabilidad y optimización de sistemas de tiempo real distribuidos generales y también basados en particionado temporal, que podrá ser aplicada en distintos entornos en los que los sistemas ciberfísicos juegan un rol clave, por ejemplo en aplicaciones de Industria 4.0, en los que pueden presentarse problemas similares en el futuro.

The increasing complexity of modern control systems leads many companies to have to resize or redesign their solutions to adapt them to new functionalities and requirements. A paradigmatic case of this situation has occurred in the railway sector, where the implementation of signaling applications has been carried out using traditional techniques that, although they currently meet the basic requirements, their time performance and functional scalability can be substantially improved. From the solutions proposed in this thesis, besides contributing to the assessment of systems that require functional safety certification, the base technology for schedulability analysis and optimization of general as well as time-partitioned distributed real-time systems will be derived, which can be applied in different environments where cyber-physical systems play a key role, for example in Industry 4.0 applications, where similar problems may arise in the future.

UC UNIVERSIDAD DE CANTABRIA

Escuela de Doctorado